(54) System for securely accessing data from smart cards

(57) A card-enabled processing system comprises a security module (120) for securely exchanging data with cards, such as smart cards, and an application module (150) for processing data from the smart cards. The security module (120) encrypts and decrypts data using keys, which are securely stored in a secure mem- ory. The security module (120) also validates the cards before processing by the application module (150) occurs and assists the card in validating the system. The application module provides a common platform in which different types of smart cards can be processed.
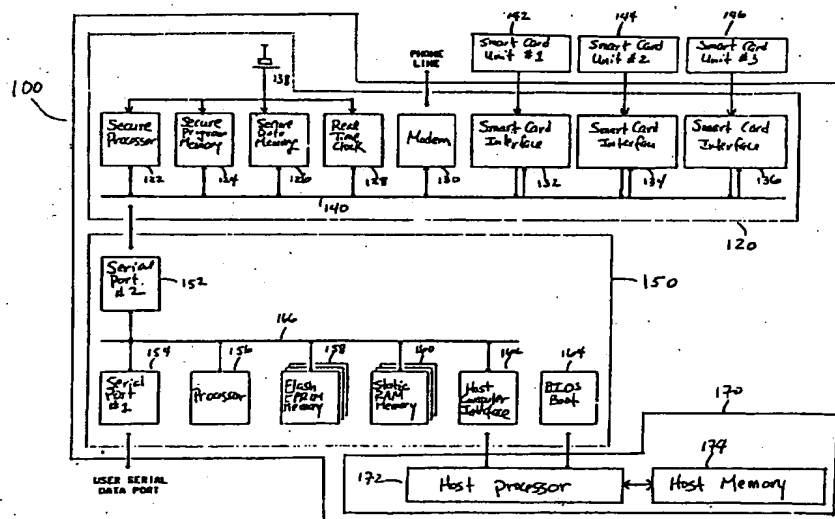
Figure 1

Description

BACKGROUND OF THE INVENTION

5 Field of the Invention

The invention generally relates to systems for providing a secure processing environment. In particular, this invention relates to controllers used in card-enabled processing systems for providing secure access to data stored in data-carrying cards, including data received from smart cards, and for providing secure communications of processed data.

10

Description of the Related Art

Business, social, and government interactions in a technological society rely increasingly on electronically stored, processed, and accessed data. For example, data-carrying cards, such as "smart" cards having internal data process-
15 ing capabilities, are used to store and process various types of data, like financial information, medical information, and welfare benefits information. Smart cards can store data relating to a transaction (e.g., a credit or a debit), eliminating the need for paper records. The use of such cards, however, raises several concerns.

One concern involves data security. Where data is sensitive or confidential, access to data stored on cards should be limited to those people with the appropriate authorization. Also, because data transmitted to and from cards can be
20 easily duplicated or altered, data authentication is desirable to prevent fraudulent use of duplicated or altered data. To further prevent data manipulation and repudiation, data certification, which is a process of providing cryptographic proof of the origin and integrity of data, can be employed.

Another concern arises from the differences between smart cards' chip operating system (COS). Lack of clear industry standards in the smart card field results in incompatibilities between smart cards produced by different manu-
25 facturers. One manufacturer's smart card processing system often does not support another manufacturer's smart cards. This results in great inconvenience to smart card users, who are often unable to use their smart cards when they cannot locate systems capable of supporting their cards. Thus, there is a need for controllably bridging the incompatibilities of such systems. For example, if a user holds a debit card and wants to have his account debited immediately for a purchase, it would be desirable for any merchant who wants to make the sale to be able to accept and employ the
30 user's debit card for that purpose, and to receive an immediate credit in his account. It would also be desirable for that merchant to be able to accept many other types of cards, such as credit cards and prepaid cards.

Despite the clarity of these needs, early attempts have proven to be inadequate. On the one hand, conventional systems have not provided adequate data security. A typical system aimed at providing a new service has a very limited environment and leaves an open end in terms of security. At present, one way that security is provided in smart cards
35 systems is to implement security features as an integrated part of the systems' application. This tactic is inefficient and inherently dangerous--inefficient because of the effort associated with software development and the maintenance costs, and dangerous because application programmers still have access to the application program that provides the system security.

On the other hand, the broad-based usability that is desired for particular applications still eludes most of the likely
40 users because of system differences large and small. Card-based systems from telephone to general credit to banking systems cannot accept cards not intended for the respective system, even when it would be to the economic advantage of the relevant business to do so. And despite the long-term need of the financial communities for a broad-based debit card, such a card and its touted advantage of a "cashless society" appears to be receding into the future. This state of events has occurred despite the proliferation of credit as a way of life and the proliferation of insertable cards as both
45 data carriers and as "smart cards," cards which both carry data and perform some on-card processing of the data.

All of the foregoing problems have led to an over-arching need, particularly felt in the business community, to address security and the incompatibility of systems concurrently, and preferably as simply as possible. The need also extends beyond the business community, for example, to government. Even though the problems in the government area differ in detail, the same problems of security and incompatibility of systems are demanding attention.

50

SUMMARY OF THE INVENTION

Accordingly, the present invention is directed to a controller that substantially obviates one or more of the problems due to limitations and disadvantages of the related art.
55 In accordance with the purpose of the invention, as embodied and broadly described, the invention includes a system for controlling requests from portable electronic cards of differing in-card electronic processing capabilities, and including a central processing module, associated memory, and an operating system, comprising a plurality of card units into which data cards and cards having internal data processing may be inserted; security module means for authenticating cards inserted into the plurality of card units and for securely exchanging data with the authenticated

cards; and means, separate from the security module means, for processing data received from the security module in accordance with an application program.

In another aspect, the invention includes a method of securely exchanging data between a data-carrying card and a processing system, comprising the steps of validating the authenticity of a data-carrying card at a security module of the processing system; providing data from the authenticated data-carrying card to the security module of the processing system; processing the data at the security module; providing the processed data from the security module to an application module; and processing the data at the application module.

In yet another aspect, the invention includes a system for providing secure exchange of data with data-carrying cards, comprising a first processor programmed to encrypt and decrypt data, the first processor being a secure processor; a secure memory connected to the secure processor for storing a security program and encryption and decryption keys; a second processor programmed to execute an application program in accordance with data received from the first processor.

In still another aspect, the invention includes a security module, comprising an input/output interface from which data can be received and transmitted; data-encryption means for encrypting data in accordance with an encryption technique using encryption keys; a memory for securely storing the encryption keys; and means for securely managing the encryption keys stored in the memory.

Additional features and advantages of the invention will be set forth in the description that follows, and in part will be apparent from the description, or may be learned by practice of the invention. The advantages of the invention will be realized by the systems and related methods of use particularly pointed out in the written description and claims hereof as well as the appended drawings.

It is to be understood that both the foregoing general description and the following detailed description are exemplary and are intended to provide further explanation of the invention as claimed.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings are included to provide a further understanding of the invention and are incorporated in and constitute a part of the specification and together with the description serve to explain the principles of the invention.

In the drawings,

Fig. 1 is a block diagram of a card-enabled processing system having a security module and an application module in accordance with a preferred embodiment of the invention;

Fig. 2 is a block diagram illustrating circuitry of a smart card unit used in the system of Fig. 1;

Fig. 3 is a block diagram depicting exemplary routines of a security program, in accordance with one embodiment of the invention; and

Fig. 4 illustrates a conceptual view of the programs executed in accordance with a preferred embodiment of the invention.

DETAILED DESCRIPTION

The card-enabled processing system of the invention comprises at least a security module for providing secure exchange of data with data-carrying cards, including smart cards, and an application module for processing the data accessed by the security module in accordance with an application program(s). The security module controls the flow of data between the system and the data-carrying cards. The application module includes a card application programming interface, which allows access to cards issued by different manufacturers.

Fig. 1 illustrates a block diagram of a card-enabled processing system 100 including security module 120, application module 150, and host computer 170. Application module 150 is connected to security module 120 and host computer 170. In a preferred embodiment, security module 120 and application module 150 comprise programmable printed circuit boards, which are inserted into slots (not shown) of host computer 170.

As shown in Fig. 1, in a preferred embodiment, security module 120 includes secure processor 122, secure program memory 124, secure data memory 126, real time clock 128, modem 130, smart card interfaces 132, 134, 136, and back-up battery 138. Processor 122, memories 124, 126, clock 128, modem 130 and interfaces 132, 134, 136 are all connected to bus 140. Back-up battery 138 is connected to processor 122, memories 124, 126, and clock 128 and acts as an independent power source for these components.

Security module 120 generally provides secure access to data stored on data-carrying cards, including data from "smart cards," which have internal data processing capabilities. For example, security module 120 provides authorization verification to cards, authenticates cards inserted into system 100, validates the integrity of data from the cards, encrypts and decrypts data accessed to and from the cards, and provides secure storage and management of authentication, encryption, and certification "keys" used to encrypt and decrypt the data.

Secure processor 122 executes software (referred to herein as "the security program") stored in secure program memory 124. The functionality provided by processor 122 executing the security program is described in greater detail below in connection with Fig. 3. Processor 122 preferably contains a Data Encryption Standard (DES) engine for encrypting data to be stored on the cards and decrypting data received from the cards. Known encryption algorithms,
5 such as Electronic Code Book (ECB) or Cipher Bloc Chaining (CBC), can also be used. These algorithms uniquely encrypt data based upon "encryption keys" (e.g., 64-bit numbers). Public key algorithms, such as digital signature algorithm (DSA) and RSA can also be used.

The encryption keys used to encrypt data received from smart cards are stored in secure data memory 126 and are preferably only accessible by processor 122. In one embodiment, the encryption keys themselves are encrypted by
10 processor 122 before they are stored in memory 126. Processor 122 can request a key from memory 126, for example, by transmitting an encrypted index, or address, to memory 126. Memory 126 receives the encrypted address, decrypts the address, finds the key stored in that address, and transmits the key to processor 122.

To ensure the security of the data received from the smart cards, secure processor 122, secure program memory 124, and secure data memory 126 are preferably designed such that any physical tampering causes the contents of
15 these devices to be erased. In a preferred embodiment, processor 122 comprises a DT2252 secure microprocessor with its associated program and data memories 124 and 126.

Real time clock 128 maintains the current date and time and preferably comprises a quartz-based clock. When directed by the security program, processor 122 receives the date and time from clock 128 and stores them on the smart cards. For instance, when a charge has been made on a credit card, the amount of the charge can be stored on
20 the credit card along with the date and time of the charge.

As shown in Fig. 1, modem 130 is connected to an external phone line. Model 130 interfaces with system 100 via an interface (not shown), such as a Signectics UART SC68C94 quad chip. Modem 130 allows remotely-located host systems, which may also be responsible for key distribution and key management, to communicate with system 100 through security module 120. Data received from modem 130 is directed to processor 122, which authenticates the
25 data in accordance with the security program. Modem 130 can be any commercially-available modem and preferably uses the Hayes AT command set. An example of such a modem is the Cermetek CH1782 2400 baud modem.

Smart card interfaces 132, 134, 136 interface smart card units 142, 144, 146, respectively, with system 100. Preferably, these interfaces 132, 134, 136 are connected to bus 140 via, for example, the Signectics UART SC68C94 quad chip. Units 142, 144, 146 accept manual insertion of the cards and establish direct electrical contact with the cards.
30 Units 142, 144, 146 read data from the cards to the respective interfaces and write data from the respective interfaces to the cards. While card units 142, 144, 146 are shown in Fig. 1 to reside outside system 100, they can alternatively be mounted in security module 120 as part of system 100.

It is noted that units 142, 144, 146 are connected to system 100 via security module 120 only. In this way, data accessed from cards inserted in units 142, 144, 146 can be verified and, if necessary, decrypted by the security pro-
35 gram before processing by application module 150 occurs. Also, data from application module 150 can be verified and encrypted by the security program before being stored on these cards.

Application module 150 receives data from security module 120 and processes this data. One aspect of application module 150 is its adaptability to a wide-variety of different data carrying and data processing-type cards. As shown in Fig. 1, application module 150 comprises serial ports 152, 154, processor 156, EPROM 158, RAM 160, host computer
40 interface 162, and BIOS boot 164, all of which are connected to bus 166.

Serial port 152 interfaces application module 150 with security module 120, allowing data from security module 120 to be transmitted to application module 150 and vice-versa.

Serial port 154 allows other systems running data gathering applications, to communicate with application module 150. Preferably, serial port 154 comprises a standard RS-232 port, to which compatible serial data ports of other sys-
45 tems can be connected via cable. For example, a transaction collection platform can be linked to system 100 through serial port 154, allowing system 100 to receive, process, and transmit transaction data to a host system via modem 130.

Processor 156 executes programs stored in EPROM 158 and stores data in RAM 160. Processor 156 preferably comprises a 80C186EB PC CPU, which can run with a 16 MHz or 32 MHz oscillator clock (not shown). The programs stored in EPROM 158 can include a "card application programming interface" (CAPI™), which allows system 100 to
50 read data from and write data to most types of smart cards, and application program(s), which define how the data from the card is to be processed. Examples of application programs include programs that maintain medical histories and/or welfare benefit histories, programs that credit charges to accounts, and programs that debit charges from accounts. The interaction between CAPI and the application program is described below in connection with Fig. 4.

As discussed above, application module 150 preferably comprises a programmable printed circuit board and has a
55 parallel 64 pin, card edge connector (not shown), which is inserted into a slot (not shown) of host computer 170. Module 150 communicates to the host computer 170 through the connector via host computer interface 162. Interface 162 is preferably a conventional interface capable of generating processor interrupts for processor 156 based upon a buffer status, such as the Am4701 FIFO Interface.

BIOS boot 164 provides boot time access control and provides an interface between input/output devices of host

computer 170, such as a host monitor (not shown) and a keypad (not shown), CAPI, and security module 120, for example, to validate personal identification numbers (PINs).

Host computer 170 includes host processor 172 and host memory 174, which are connected to each other as shown in Fig. 1. In a preferred embodiment, host computer 170 comprises a personal computer, such as the IBM PC or suitable compatible, having a plurality of slots in which security module 120 and application module 150 can be inserted. Host computer 170 also preferably includes input/output devices, such as a keypad (not shown) and a display (not shown), to permit a user to receive and input information into system 100.

Host processor 172 can execute an operating environment, such as an operating system, which is independent of the application programs executed by processor 156 and the security program stored in program memory 124. Host processor 172 also preferably executes software that controls the input/output devices of host computer 170, including interface programs for prompting the user to enter desired information, such as a PIN.

Although the preferred embodiment of the invention has been described as including an application module having its own processor for executing application programs independent of the host processor, the application programs described above could alternatively be executed directly by the host processor 172. Similarly, the processor of the application module could execute some or all of the functionality described as being implemented by the host processor.

Fig. 2 illustrates circuitry of a card unit 200 which may be used for card units 142, 144, 146 shown as blocks in Fig. 1. Card unit 200 includes contact unit 202 which has a slot for insertion of a portable data-carrying card or smart card, control transistor 204 which controls whether the reader is on or off in accordance with a "reader on/off" signal, and analog multiplexer 206 for switching between a synchronous clock and an asynchronous clock, so that multitude of data carrying cards (e.g., smart cards) can be accepted by security module 120. Contact unit 202 receives data from and sends data to a card interface over a "card data" line 208.

Fig. 3 illustrates a block diagram depicting exemplary routines of the security program executed by processor 122, in accordance with one embodiment of the invention. The security program includes Authenticate Card Routine 300, Authenticate Host Routine 310, Encryption Routine 320, Decryption Routine 330, Authenticate Message Routine 340, Key Management Routine 350, and Verify Authorization Routine 360, and Certificate Generation Routine 370. These routines are executed by processor 122, preferably when and as needed to secure data handled by the application program stored in memory 158.

Authenticate Card Routine 300 is provided to allow system 100 to determine whether a card inserted into one of the card units is authentic (e.g., issued by an authorized institution). Under routine 300, processor 122 generates a random number and transmits it to the card. The card receives the random number, encrypts the number based upon an algorithm and an "internal key" stored in the card, and returns the encrypted random number to processor 122. The internal key uniquely identifies the card as being an authentic card. Processor 122 decrypts the number based upon the same algorithm and an identifying key stored in memory 126. Processor 122 compares the original random number and the decrypted random number. If they are the same, processor 122 determines that the card is authentic. In contrast, if they are different, processor 122 determines that the card is not authentic.

Authenticate Host Routine 310 is provided to allow a card to determine whether the processing system in which the card is inserted is authentic. Under routine 310, the card generates a random number and transmits it to processor 122. Processor 122 receives the random number, encrypts the random number based upon an algorithm and an identifying key stored in memory 126, and returns the encrypted random number to the card. The card decrypts the received number from processor 122 based upon the same algorithm and an internal key stored in the card. Again, the internal key uniquely identifies the card as being authentic. The card compares the original random number with the decrypted random number. If they are the same, the card determines that system 100 is authentic. In contrast, if they are different, the card determines that system 100 is not authentic.

As their names suggest, encryption routine 320 encrypts data and decryption routine 330 decrypts data. These routines are executed when data is transmitted from or received by system 100, and are particularly valuable for data that is sensitive or confidential. As discussed above, the encryption and decryption algorithms employed by system 100 use encryption and decryption keys to uniquely encrypt and decrypt data, respectively. Known algorithms may be used, including "symmetrical" algorithms (e.g., decryption keys are derivable from encryption keys), such as DES, ECB, or CBC, and "asymmetrical" algorithms, such as DSA and RSA. Although data encrypted using asymmetrical algorithms is generally more secure than data encrypted using symmetrical algorithms, symmetrical algorithms are often adequate. As discussed above, the keys are securely stored in memory 126.

Authenticate Message Routine 340 allows a card to determine whether data from system 100 is authentic. Secure processor 122 generates a message authentication code (MAC) from data that processor 122 transmitted to the card based upon an algorithm and a MAC key stored in memory 126 and transmits the MAC to the card. The card generates its own MAC from the data it received from processor 122 based upon the same algorithm and a MAC key stored in the card. The card compares the MAC generated by the card and the MAC generated by the processor 122. If they are the same, the card determines that the data is authentic. In contrast, if they are different, the card determines that the data is not authentic. This routine 340 can also operate to allow processor 122 to determine whether data received from a card is authentic. In that case, processor 122 compares the MACs generated by processor 122 and the card and makes

a determination of whether the data from the card is authentic.

Key management routine 350 is responsible for maintaining, creating, issuing, changing, setting, storing, and retrieving encryption and decryption keys stored in memory 126. Routine 350 creates encryption keys for new application programs as they are required. Preferably, each application program is only allowed to have a set number of encryption keys corresponding to that application program. During execution of an application program, routine 350 can set only one key to be active at a time. Thus, at any given time, the key currently set by routine 350 is the only one that is available.

When processor 122 retrieves encryption and/or decryption keys from memory 126 in accordance with routine 350, processor 122 transmits an index, preferably encrypted, which corresponds to an address in memory 126. The key associated with that index, or address, is then returned.

Verify Authorization Routine 360 determines whether the user is authorized to use the card inserted into system 100. Under routine 360, system 100 prompts the user to enter a PIN. Processor 122 encrypts the PIN and transmits it to the card. The card decrypts the received PIN and compares it with a PIN stored in the card. If the PINs match, then system 100 determines that the user is authorized and allows the user to gain entry into system 100. However, if the PINs do not match, then system 100 determines that the user is not authorized to use the card.

Certificate Generation Routine 370 generates "certificates" or "signatures" from blocks of data and appends the certificates or signatures to the blocks of data. Preferably, in generating the certificates or signatures, routine 370 processes the blocks of data using an algorithm and a certification key stored in memory 126. For each block of data, routine 370 appends the generated certificate or signature to the block of data. In this way, the integrity of the data blocks can be maintained by reconstructing the blocks of data from the certificates or signatures, if necessary. Also, using the certificates or signatures, system 100 can provide non-repudiation of the blocks of data.

In a preferred embodiment, routines 300, 310, 320, 330, 340, 350, 360, 370 comprise software routines, such as microcode, executed by processor 122 and any hardware necessary to effect the execution of that microcode in accordance with conventional techniques to perform the specified functionality. In an alternative embodiment, this functionality can be implemented solely in hardware (e.g. electronic circuitry) in accordance with conventional techniques, or partly in software and partly in hardware.

In alternative embodiments of the security program, other routines can be provided in the security program. These routines may provide different security functions from those described above. For example, the Appendix attached hereto lists and describes routines which can be included in the security module and depicts these routines as flow diagrams.

Fig. 4 provides a conceptual view of the programs executed by processor 156, including application program 400 and CAPI 410 (CAPI™ stands for card application program interface). As shown in Fig. 4, CAPI includes application layer 412, card layer 414, and International Standards Organization (ISO) layer 416. This structure provides flexibility during execution, for example, by allowing data from application program 400 to flow directly to the application layer 412 or directly to card layer 414 or directly to the ISO layer 416. In addition, certain data from the application layer 412 can pass serially through the subordinate functions represented by card layer 414 and the International Standards Organization Layer 416.

Application program 400 resides in the host 170 and/or the application module 150, or is imported from the card by application module 150. Program 400 processes data from smart cards and performs appropriate transactions with that data. During execution of application program 400, data can be read from or stored on smart cards inserted into system 100. Various different types of application programs can be implemented. For instance, application program 400 can be a debit program which debits charges from a user's bank account. In this case, as a preliminary matter, the application program 400 preferably commands the security module to verify use of the card by executing the verify authorization routine 360, the authenticate card routine 300, and the authenticate host routine 310.

If verification is successful, application program 400 causes processor 156 to receive debit information from the card, such as an account number, maximum debit per day, maximum debit per transaction, and total debit per day, through CAPI. Under control of program 400, processor 156 communicates with a system belonging to the user's bank (not shown) via serial port 154, or alternatively modem 130, to verify that the user's account contains sufficient funds to cover the charge. If so, then processor 156 determines whether the charge is proper (e.g., the charge is below the maximum debit per transaction) and transmits an appropriate request (e.g., if proper, charge the account). Processor 156 transmits transaction data to the inserted card for storage therein via the security module and concludes the session between the card and system 100. Alternatively, serial port 154 and modem 130 can also be used to download transaction data to the remote system. In addition, processor 122 or 156 can execute proprietary host communication protocols to exchange data and secret keys, such as encryption keys.

Numerous different types of application programs can be provided for application program 400, such as credit transaction programs, prepaid-card programs, medical history programs, and welfare benefit programs. Preferably, multiple application programs are stored in memory 156 so as to support a wider variety of card types (e.g., debit cards, credit cards, prepaid cards). If none of the application programs provided in memory 156 support the type of card inserted into system 100, then system 100 informs the user that the user's card cannot be processed by this system. In

another embodiment, the application program can be read in from the inserted card, subject to verifications by the security module 120.

CAPI 410 allows application programs to communicate with different types of cards using different protocols without the need for the application programs to be card specific. Application layer 412 is the primary interface between the cards and the application programs and provides management of the smart card environment through simplified industry specific tool sets. The card layer 414 provides direct access to the smart card functions. ISO layer 416 controls system functions of the card units, which functions have already been conformed or established as industry standards. Since a smart card is a computer with its own chip operating system (COS), when the card is powered on or reset, the card returns an "ANSWER-TO-RESET" (ATR) to a terminal (e.g., card reader). Card layer 414 processes the ATR and translates application layer 412 calls into card dependent commands, and passes the commands to the ISO layer 416. Hence, the application remains transparent to changes of card or system. CAPI allows applications to remain the same for different card manufacturers or COSs.

Software routines which can be included in the application layer, the card layer, and the ISO layer are listed and depicted as flow diagrams in the attached Appendix.

Accordingly, in accordance with the invention, the security module serves as an intermediary between smart cards and the application module, thereby ensuring a secure environment in the system. Also, because the application program running in the host 170 does not provide system security, the risk that application programmers will be able to access the system 100 is minimized.

APPENDIX

Host System

Security Program

ConnectToSecurityModule ()
DisconnectSecurityModule ()
VerifyDebitCertificate ()
VerifyCreditCertificate ()
CreateKey ()
DeleteKey()
SelectKey ()
DiversifyKey ()
SelectDiversifiedKey ()
Encrypt ()
Decrypt ()
CreateSecurityModuleUser ()
DeleteSecurityModuleUser ()
PresentSecurityModuleUser ()
UpdateSecurityModuleUser ()

ISO Layer

ConnectToReaderReq ()
DisconnectReaderReq ()
PowerOnCardReq ()
PowerOffCardReq ()
ISOINReq ()
ISOOUTReq ()
GeneralCommandReq ()

CAPI

Card Layer

CreditReq ()
CheckCodeReq ()
CheckKeyReq ()
DebitReq ()
GetResponseReq ()
InvalidateKeyReq ()
ReadBalanceReq ()
ReadFileReq ()
ReadMemoryReq ()
SelectPurseKeyReq ()
SetOptionsReq ()
SignReq ()
SubstituteDebitReq ()
UpdateCodeReq ()
UpdateFileReq ()
WriteMemoryReq ()

Application Layer

OpenCard ()
ReadCardSerialNumber ()
CloseCard ()
CreateApplication ()
OpenApplication ()
CreateUser ()
PresentUser ()
UpdateUser ()
LockUser ()
UnlockUser ()
CreateFile ()
OpenCardFile ()
ReadCardFile ()
UpdateCardFile ()
CreatePurse ()
OpenPurse ()
ReadPurse ()
DebitPurse ()
CreditPurse ()
SignTransaction ()
DisablePurse ()

ConnectToReader( ) Flow Chart

"CHART1 VSD

DisconnectToReader( ) Flow Chart

```
                        BEGIN

                          |
                          v
              Send Message to AC
              Server to disconnect
                    reader
                          |
                          v
                      Set Timer
                          |
                          v
              Read Message from
                    Windows
                          |
                          v
                    Message =
                  WM_DD_NACK
          Y                         N
```

```
                Message =
              WM_DDE_DATA
        Y                      N
                          |
                          v
                Message =
                WM_TIMER           N
                          Y
```

```
          Kill Timer              Kill Timer         RC= READERNOTRESPO
      Read IParam Message     Read IParam Message
              |                       |
              v                       v
      IParam Message =         RC = CAPISUCCESS
      DISCONNECTREADER
          Y                    RC = UNKNOWNERROR
          N
```

```
          Kill Timer
      Read IParam Message
              |
              v
      IParam Message =         RC = ERROR
      DISCONNECTREADER
          Y
          N                    RC = UNKNOWNERROR        END
```

"CHART2.VSD"

OpenCard( ) Flow Chart

5

10

```
        ╭─────────────╮
        │             │
        │    BEGIN     │
        │             │
        ╰─────────────╯
               │
               ▼
        ┌┬───────────────┬┐
        ││ PowerOnCardReq( ) ││
        │└               ┘│
        └─────────────────┘
               │
               │  RC = return of PowerOnCardReq
               │
               ▼
        ╭─────────────╮
        │    END       │
        ╰─────────────╯
```

15

20

25

30

35

40

45

50   "CHART3.VSD"

55

11
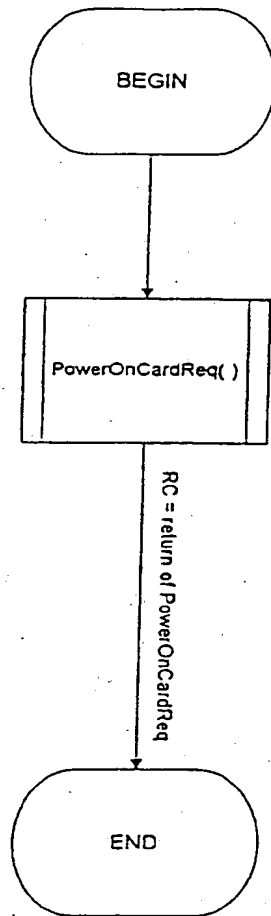
ReadCardSerialNumber( ) Flow Chart

5

10

BEGIN

15

20

PCOSReadMemory
Req( )

25

RC = return of
PCOSReadMemoryReq

30

35

END

40

45

50

"CHART4.VSD"

55

CloseCard( ) Flow Chart

5

10

15

```
      ┌─────────────┐
      │    BEGIN     │
      └─────────────┘
             │
             ▼
```

20

```
   ┌─┬─────────────────┬─┐
   │ │ PowerOffCardReq( )│ │
   │ │                   │ │
   └─┴─────────────────┴─┘
             │
             │  RC = CAPISUCCESS
             ▼
```

25

30

```
      ┌─────────────┐
      │     END      │
      └─────────────┘
```

35

40

45

50

"CHART5.VSD"

55

CreateApplication( ) Flow Chart

5

10 BEGIN

15 CreatePCOSFile( )

20

25 Create first file ok? — N

Y

30 PCOSUpdateFileReq
( )

35

40 Update first file ok? — N → RC = return of PCOSUpdateFileReq

Y

45 CreatePCOSFile( )

50

Create second file ok? — N

Y

PCOSUpdateFileReq
( )

Update second file ok? — N

Y

END

RC = return of CreatePCOSFile

RC = CAPISUCCESS

RC = return of PCOSUpdateFileReq

RC = return of CreatePCOSFile

RC = return of CreatePCOSFile

"CHART6.VSD"

55

14

OpenApplication( ) Flow Chart
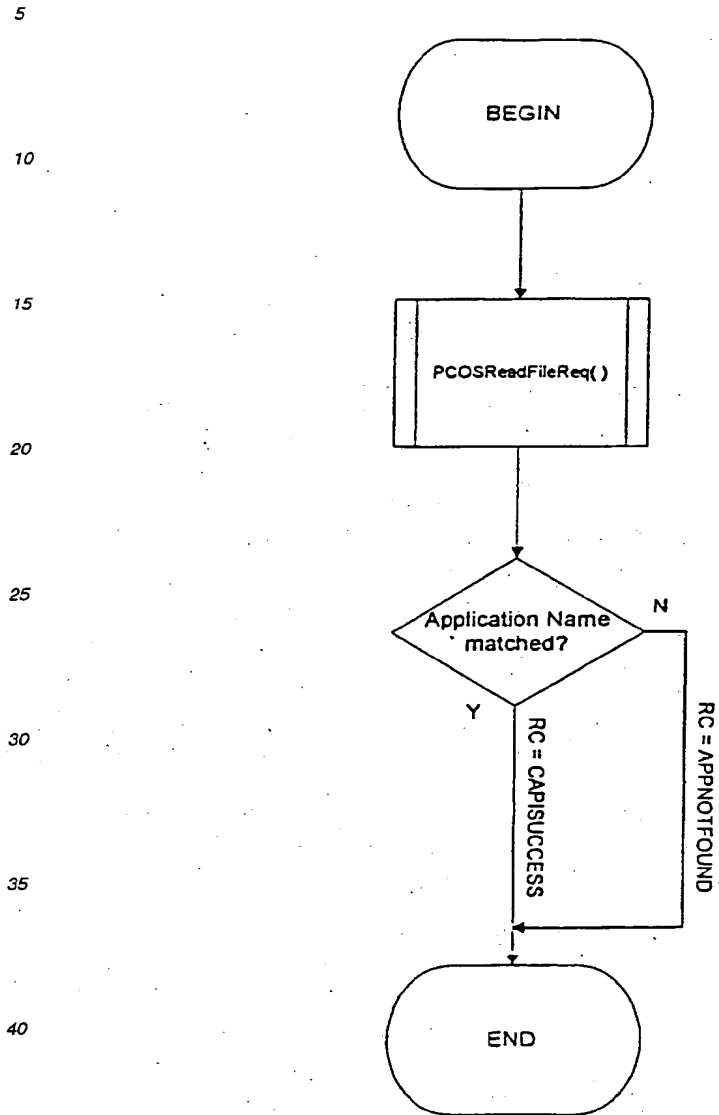
CreateUser( ) Flow Chart
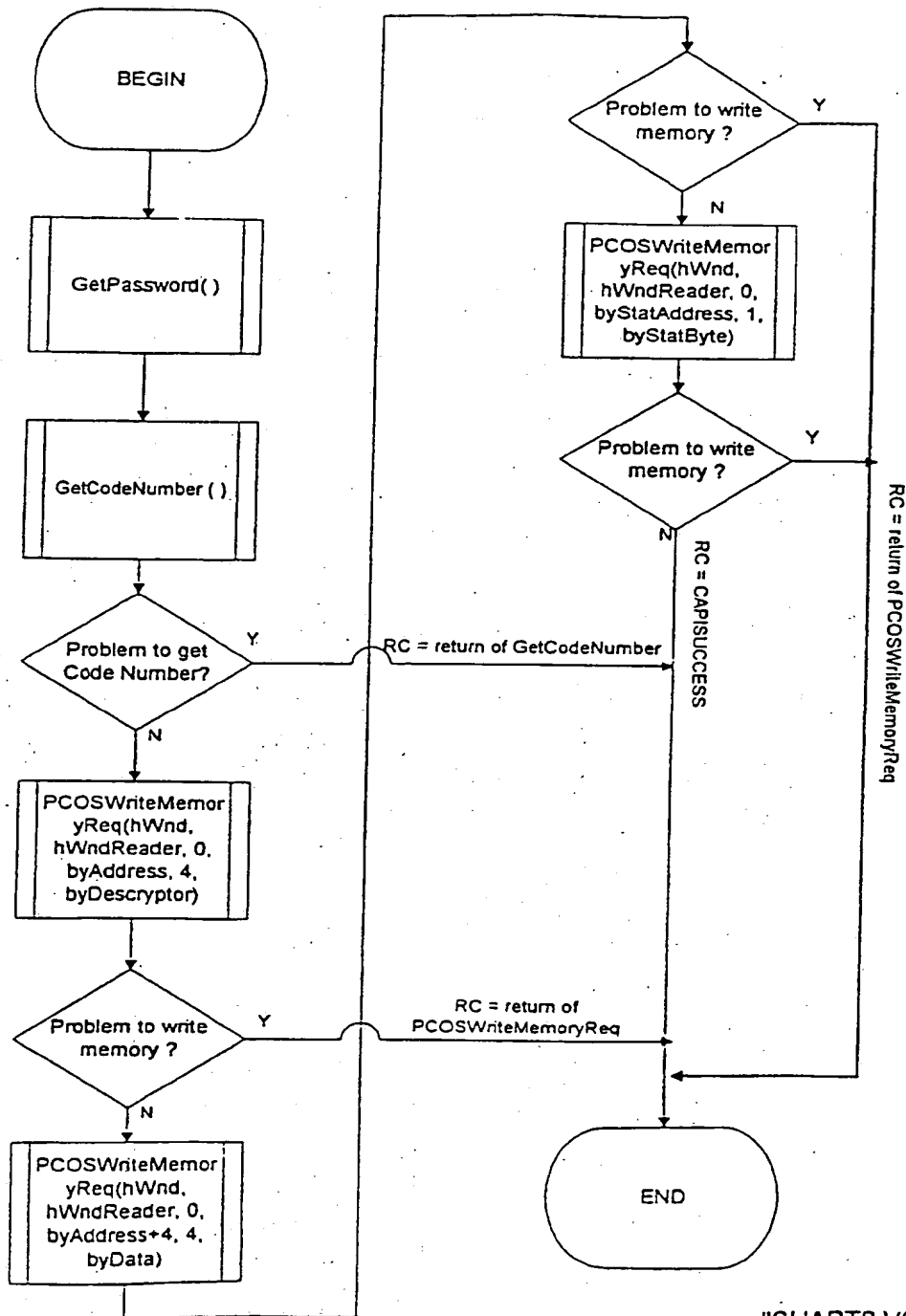
"CHART8.VSD"

PresentUser( ) Flow Chart
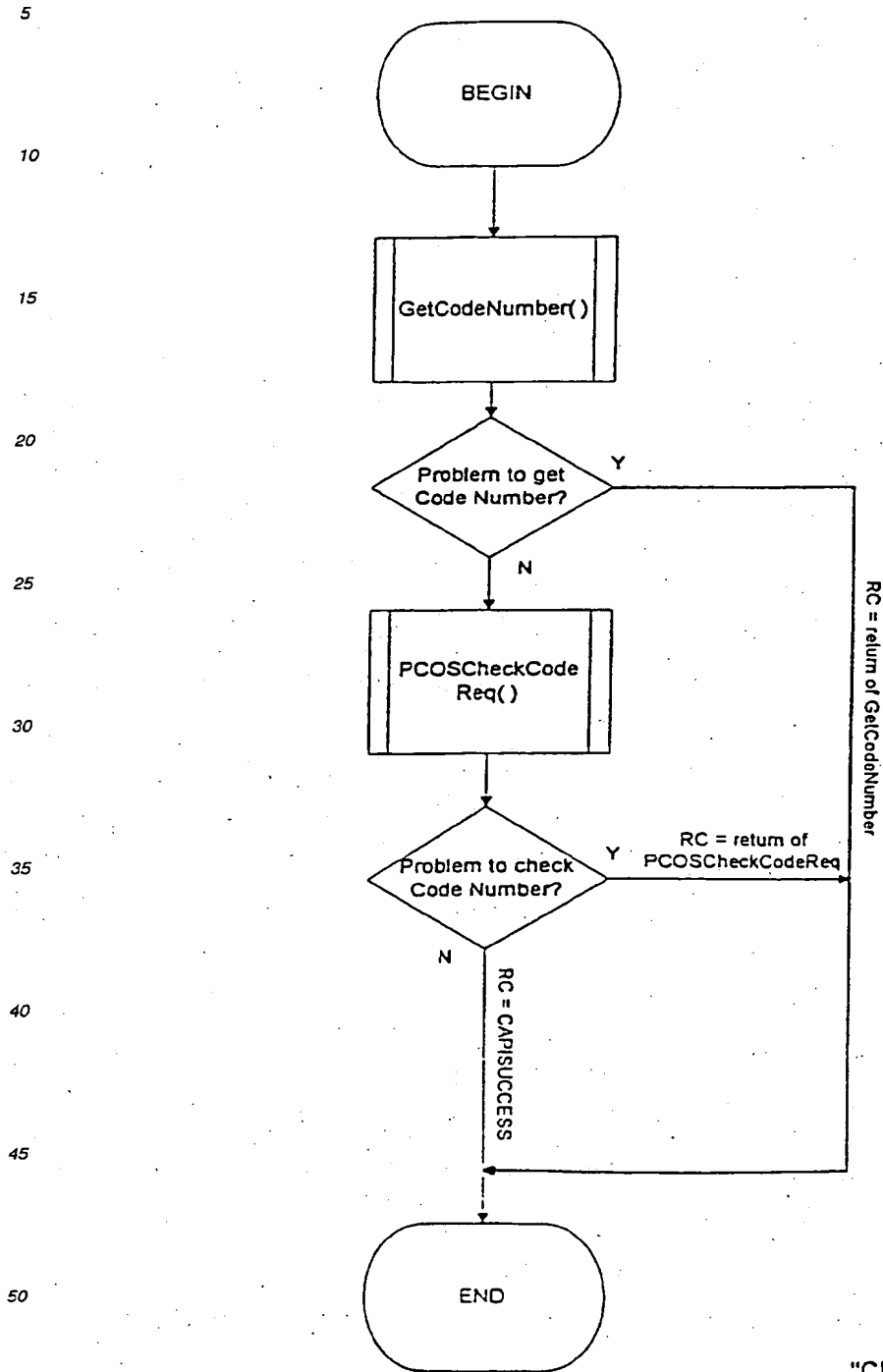
"CHART9.VSD"

UpdateUser( ) Flow Chart

CreateFile( ) Flow Chart

"CHART13.VSD"

OpenCardFile( ) Flow Chart

5

10

15

20

25

30

35

40

45

50

55

```
          ╭─────────────╮
          │    BEGIN     │
          ╰─────────────╯
                 │
                 ▼
         ┌┬─────────────┬┐
         ││ PCOSReadFileReq( ) ││
         └┴─────────────┴┘
                 │
                 ▼
            ◇ Problem to read ◇ ──Y──  RC = return of PCOSREADFILEReq ──┐
            ◇     file?      ◇                                          │
                 │ N                                                    │
                 ▼                                                      │
            ◇ if file index < ◇ ──N──  RC = FILENOTFOUND ──────────────┤
            ◇    MAXFILE     ◇                                          │
                 │ Y                                                    │
                 ▼                                                      │
          ┌─────────────┐                                              │
          │ Get file name │                                            │
          └─────────────┘                                              │
                 │                                                      │
                 ▼                                                      │
     N ──◇ if file name is ◇                                           ▼
        ◇    matched     ◇                                      ╭─────────────╮
                 │ Y                                            │     END      │
                 └── RC = CAPISUCCESS ──────────────────────────╰─────────────╯
```

"CHART14.VSD

ReadCardFile( ) Flow Chart

5

10

```
        ┌─────────────┐
        │   BEGIN      │
        └──────┬──────┘
```

15

```
        ┌──╥─────────────╥──┐
        │  ║  PCOSReadFileReq( ) ║  │
        └──╨─────────────╨──┘
```

20

25

```
              ◇ Problem to read ◇  Y
                    file?
```

RC = return of PCOSReadFileReq

30

RC = CAPISUCCESS

35

40

```
        ┌─────────────┐
        │    END       │
        └─────────────┘
```

45

50

"CHART15.VSD"

55

UpdateCardFile( ) Flow Chart

"CHART16.VSD"

Cr atePurse( ) Flow Chart

"CHART17.VSD"

A2

CreatePCOSFile( )

(Key File)

Problem to create
PCOS file? — Y — RC = return of CreatePCOSFile

N

PCOSUpdateFileReq( )

(Key File)

Problem to update
PCOS file? — Y — RC = return of PCOSUpdateFileReq

N

PCOSUpdateFileReq( )

(Purse Info File)

Problem to update
PCOS file? — Y — RC = return of PCOSUpdateFileReq

N

PCOSUpdateFileReq( )

(Purse File)

Problem to update
PCOS file? — Y — RC = return of PCOSUpdateFileReq

N

GetCodeNumber( )

Problem to get
code number? — Y — RC = return of GetCodeNumber

N

PCOSUpdateFileReq( )

(Purse File)

Problem to update
PCOS file? — Y — RC = return of
PCOSUpdateFileReq

N

PCOSUpdateFileReq( )

(Purse File)

Problem to update
PCOS file? — Y — RC = return of
PCOSUpdateFileReq

N

PCOSUpdateFileReq( )

(Purse File)

Problem to update
PCOS file? — Y — RC = return of
PCOSUpdateFileReq

N

A10

A3

A10

A10

24

A4

5

SelectKey
(hSecurityModule,
3)

10

15   Problem to select   Y   RC = return of SelectKey
      key?

N

20

Encrypt( )

25

Problem to   Y   RC = return of Encrypt
encrypt?

30

N

PCOSUpdateFileReq( )

(Purse File)

35

40   Problem to update   Y   RC = return of
      PCOS file?   PCOSUpdateFileReq

N

45

A10

SelectKey
(hSecurityModule,
4)

Problem to select   Y   RC = return of SelectKey
key?

N

Encrypt( )

Problem to   Y   RC = return of Encrypt
encrypt?

N

PCOSUpdateFileReq( )

(Purse File)

Problem to update   Y   RC = return of
PCOS file?   PCOSUpdateFileReq

N

A10

AS

50   "CHART17.VSD

55

"CHART17.VSD"

A3

SelectKey
(hSecurityModule,
1)

Problem to select
key?   Y   RC = return of SelectKey

N

Encrypt( )

Problem to
encrypt?   Y   RC = return of Encrypt

N

PCOSUpdateFileReq( )

(Purse File)

Problem to update
PCOS file?   Y   RC = return of
PCOSUpdateFileReq

N

SelectKey
(hSecurityModule,
2)

Problem to select
key?   Y   RC = return of SelectKey

N

Encrypt( )

Problem to
encrypt?   Y   RC = return of Encrypt

N

PCOSUpdateFileReq( )

(Purse File)

Problem to update
PCOS file?   Y   RC = return of
PCOSUpdateFileReq

N

A10

A4

A10

"CHART17.VSD"

OpenPurse( ) Flow Chart

BEGIN

PCOSReadFileReq( )

Problem to read file? — Y — RC = return of PCOSREADFILEReq

N

if file index < MAXFILE — N — RC = FILENOTFOUND

Y

Get file name

if file name is matched — N

Y — RC = CAPISUCCESS

END

"CHART18.VSD"

ReadPurse( ) Flow Chart

```
        ┌─────────────┐
        │    BEGIN    │
        └─────────────┘
               │
               ▼
        ┌─────────────┐                    ┌─────────────┐
        │PCOSReadFileReq()│                │PCOSRead Memory│
        │             │                    │    Req( )    │
        │ (Purse Info)│                    └─────────────┘
        └─────────────┘                           │
               │                                  ▼
               ▼                             ╱Problem to read╲  Y   RC = return of PCOSReadMemoryReq
          ╱Problem to read╲  Y              ╲   memory?    ╱──────────────────────────►
          ╲    file?     ╱──────►                 │ N
               │ N                                 ▼
               ▼                                                RC = return of PCOSREADFILEReq
        ┌─────────────┐                    ┌─────────────┐ ──────────────────────────►
        │PCOSSelectPurse│                  │SelectKey(hSecurity│
        │   KeyReq( )  │                    │  Module, 1) │
        └─────────────┘                    └─────────────┘
               │                                  │
               ▼                                  ▼        RC = return of PCOSSelectPurseKeyReq
          ╱Problem to select╲  Y                           ──────────────────────────►
          ╲  purse key?   ╱──────►
               │ N                                 ▼
               ▼                             ╱Problem to select╲  Y   RC = return of SelectKey
        ┌─────────────┐                      ╲    key?      ╱──────────────────────────►
        │PCOSGetResponse│                          │ N
        │    Req( )    │                            ▼
        └─────────────┘                                      RC = return of PCOSGetResponseReq
               │                                    ──────────────────────────►
               ▼
          ╱Problem to get╲  Y
          ╲  response?  ╱──────►
               │ N                            (A2)                        (A10)
```

"CHART19.VSD"

"CHART19.VSD

DebitPurse( ) Flow Chart

5

10

15

20

25

30

35

40

45

50

"CHART20.VSD"

55

CreditPurse( ) Flow Chart

"CHART21 VSD"

onnectToSecurityModule( ) Flow (irt

5

10

```
        ╭─────────────╮
        │    BEGIN     │
        ╰─────────────╯
               │
               ▼
15     ┌─────────────────┐
       │  Get device name │
       │  Get user name   │
       │  Get password    │
       └─────────────────┘
               │
20             ▼
       ┌─────────────────┐
       │ Initiate with SAM│
       │      Server      │
       └─────────────────┘
               │
25             ▼
            ◇─────────◇   Y   RC = NOREADERFOUND
           Timer runs out ──────────────────────┐
            ◇─────────◇                          │
30             │ N                               │
               ▼                                 │
            ◇─────────◇   Y   RC = NOREADERFOUND │
35           SAM server  ─────────────────────►  │
             acked.?                              │
            ◇─────────◇                          │
               │ N                               │
               │ RC = CAPISUCCESS                │
40             │                                 │
               │                                 │
               │                                 │
45             ▼◄────────────────────────────────┘
        ╭─────────────╮
        │     END      │
        ╰─────────────╯
```

50

"CHART24.VSD"

55

VerifyDebitCertificate( ) Flow Chart

```
                    ┌─────────────┐
                   (    BEGIN      )
                    └──────┬──────┘
                           │
                    ┌──────▼──────┐
                    │Initialize byData│
                    └──────┬──────┘
                           │
                  ┌┌──────▼──────┐┐
                  ││ DES(1, byData,││
                  ││  bySession.   ││
                  ││  byOutData)   ││
                  └└──────┬──────┘┘
                           │
                          ╱▼╲            N
                        ╱      ╲ ──────────┐
                       ⟨Cetificate matched?⟩│
                        ╲      ╱            │
                          ╲  ╱              │
                          Y│               │
                    RC = CAPISUCCESS    RC = CAPIFAIL
                           │               │
                           ◄───────────────┘
                           │
                    ┌──────▼──────┐
                   (     END       )
                    └─────────────┘
```

"CHART26.VSD"

VerifyCreditCertificate( ) Flow Chart

"CHART27.VSD"

CreateKey( ) Flow Chart
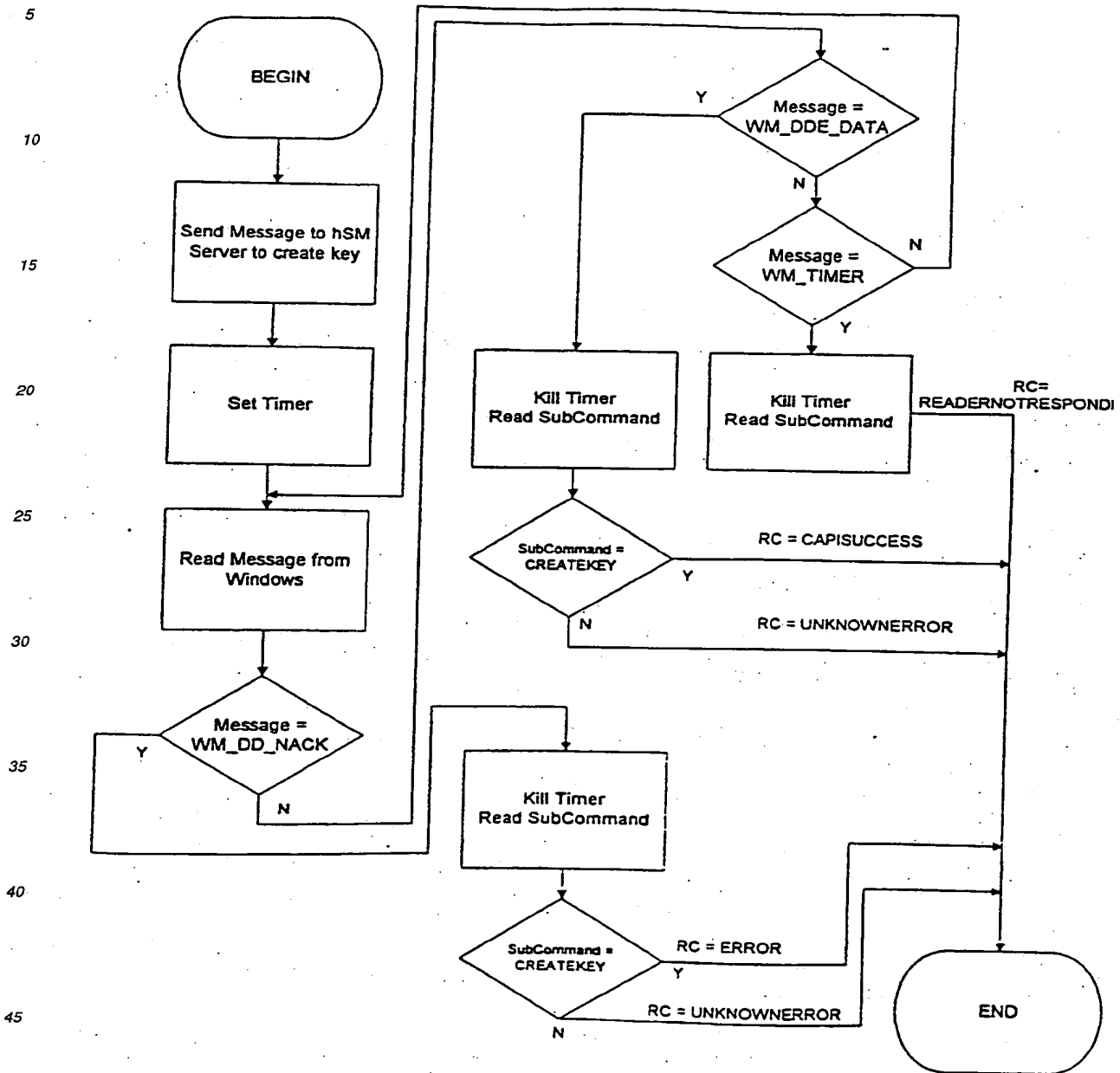
"CHART28.VSD"

DeleteKey( ) Flow Chart
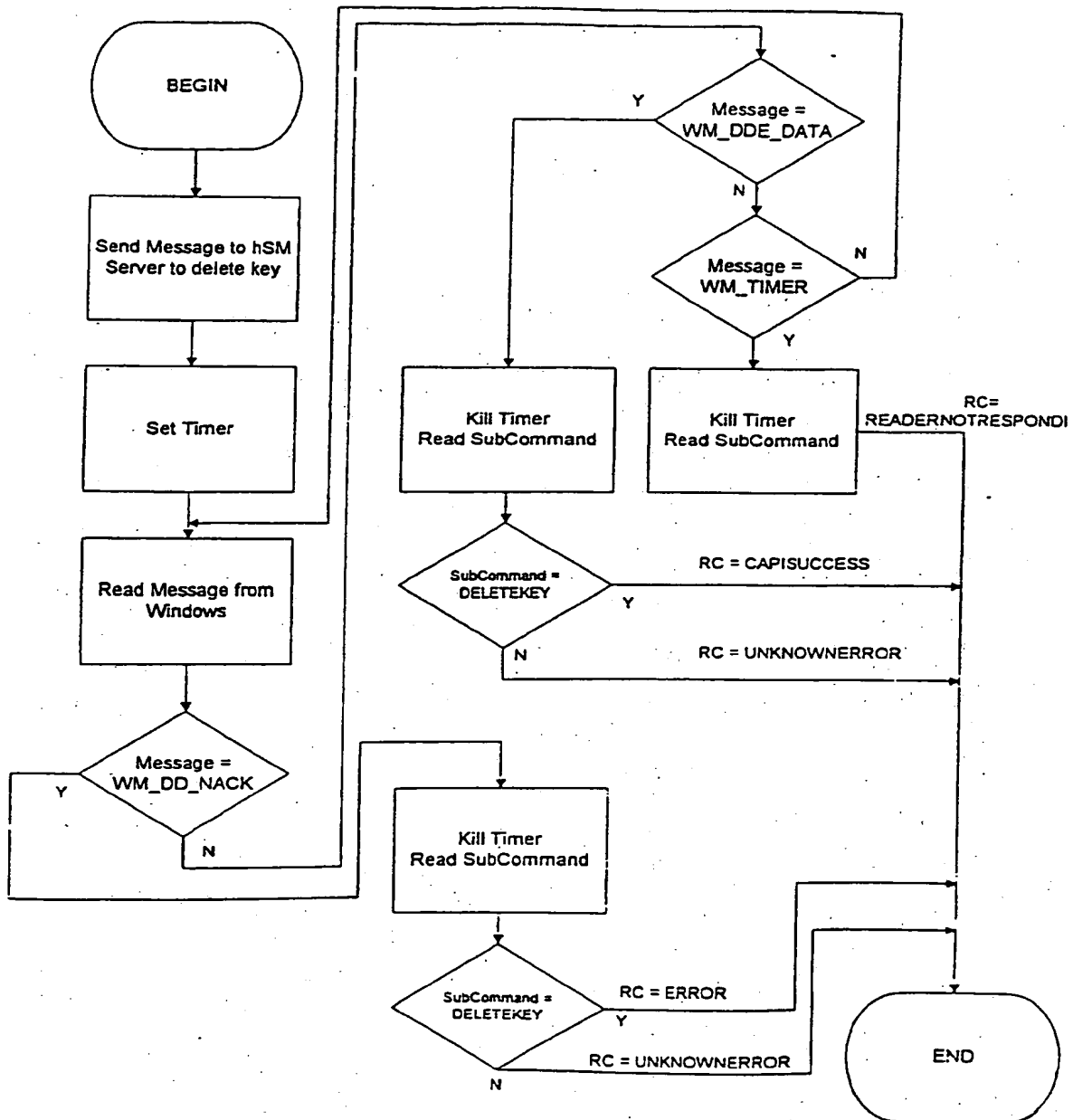
"CHART29.VSD"

SelectKey( ) Flow Chart
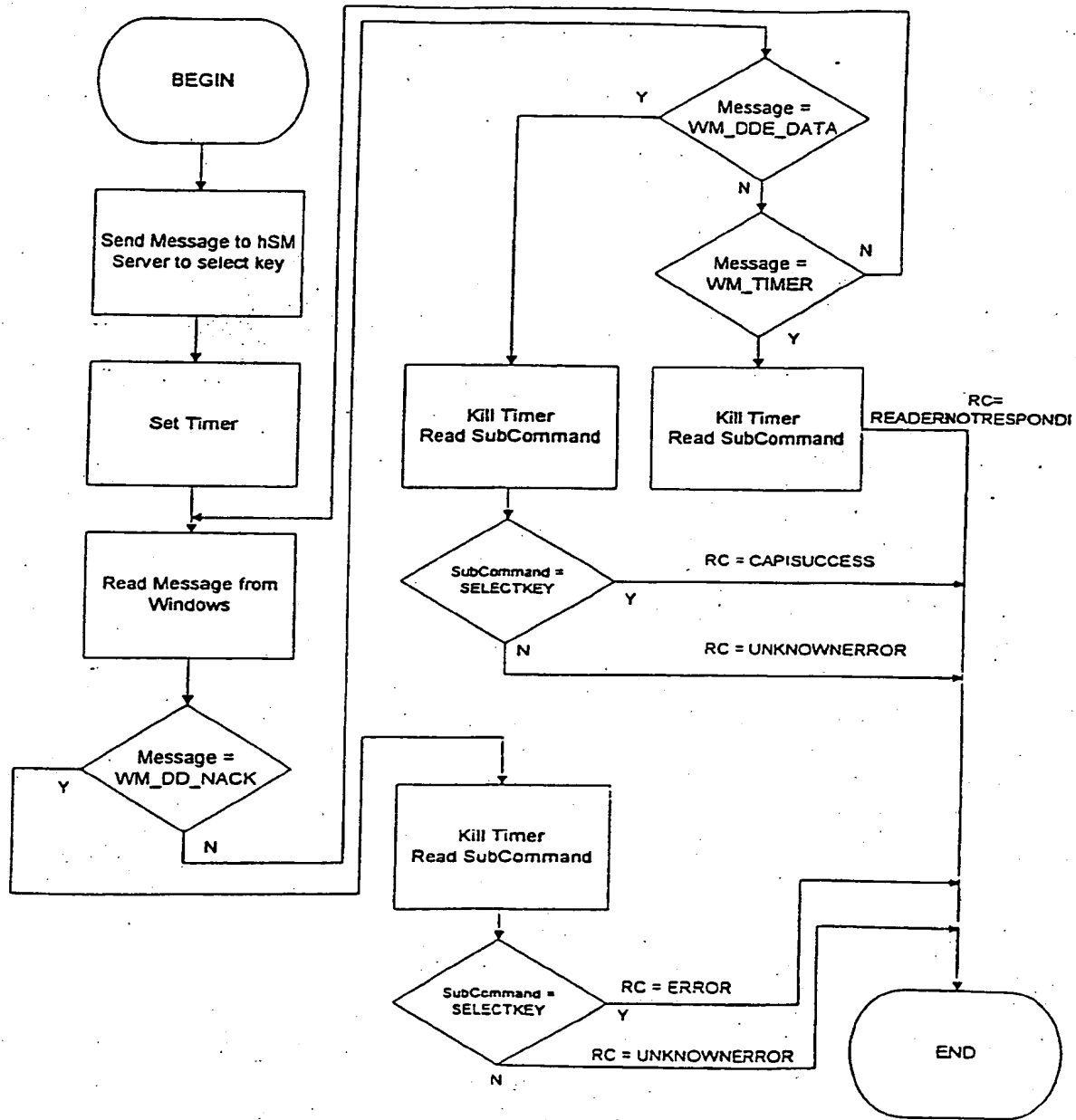
"CHART30.VSD"

DiversifyKey( ) Flow Chart

5

10

```
        ╭─────────────╮
        │    BEGIN     │
        ╰──────┬──────╯
               │
               ▼
        ┌─┬─────────────┬─┐
        │ │DiversifyKeyReq( )│ │
        │ │             │ │
        └─┴──────┬──────┴─┘
               │
               ▼
        ╭─────────────╮
        │     END      │
        ╰─────────────╯
```
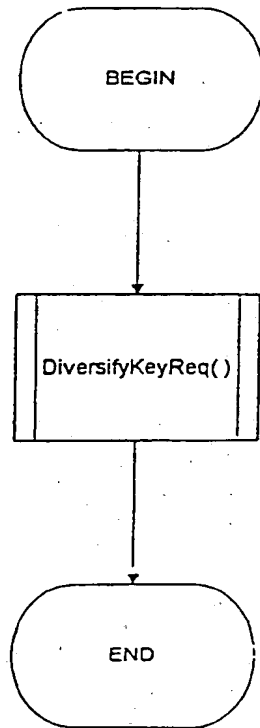
"CHART31.VSD"

Encrypt( ) Flow Chart

BEGIN

Send Message to hSM
Server to encrypt

Set Timer

Read Message from
Windows

Message =
WM_DD_NACK

Message =
WM_DDE_DATA

Message =
WM_TIMER

Kill Timer
Read SubCommand

Kill Timer
Read SubCommand

RC=
READERNOTRESPONDII

SubCommand =
ENCRYPT

RC = CAPISUCCESS

RC = UNKNOWNERROR

Kill Timer
Read SubCommand

SubCommand =
ENCRYPT

RC = ERROR

RC = UNKNOWNERROR

END

"CHART33.VSC

5

10

15

20

25

30

35

40

45

50

55

40
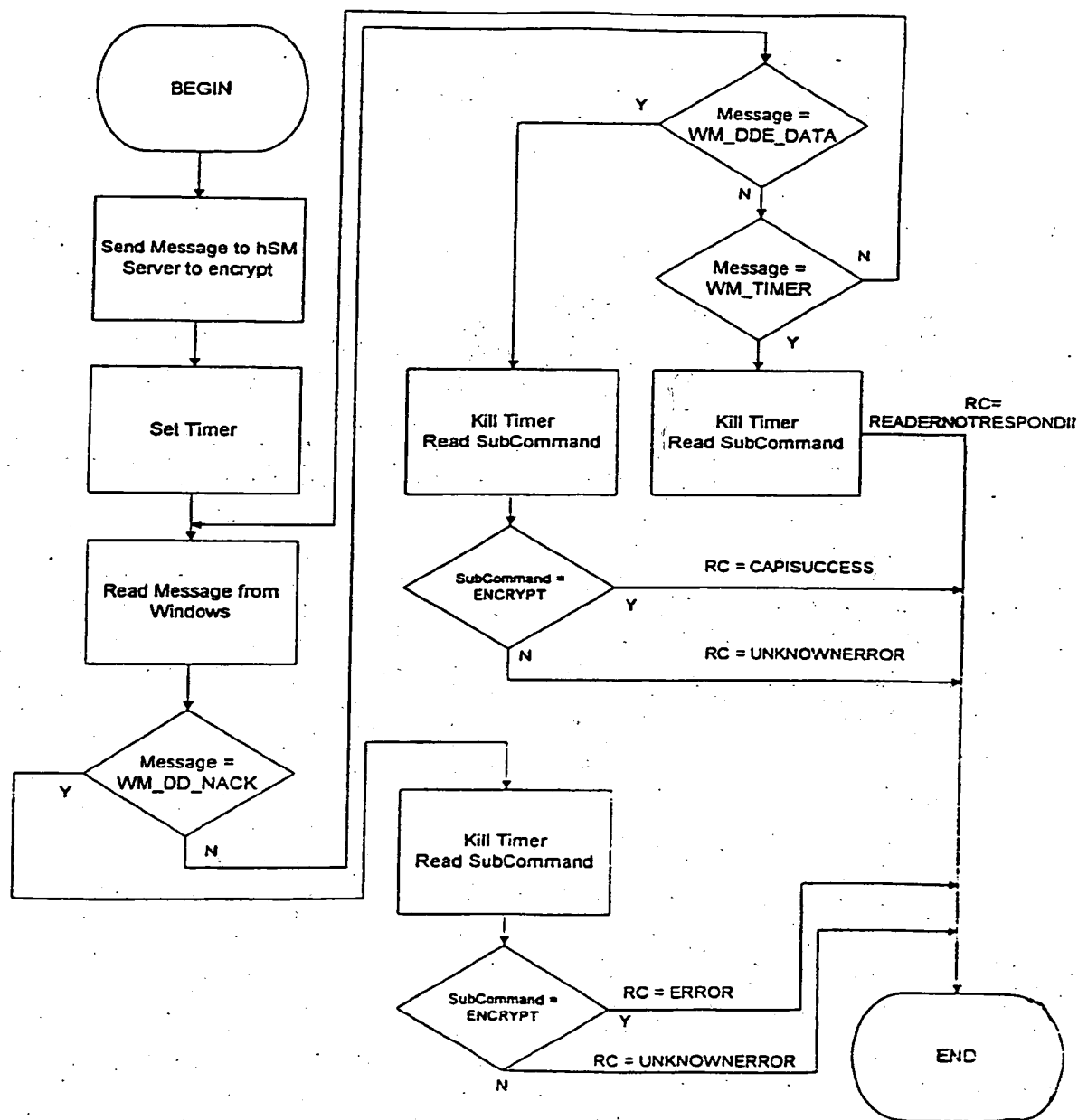
Decrypt( ) Flow Chart
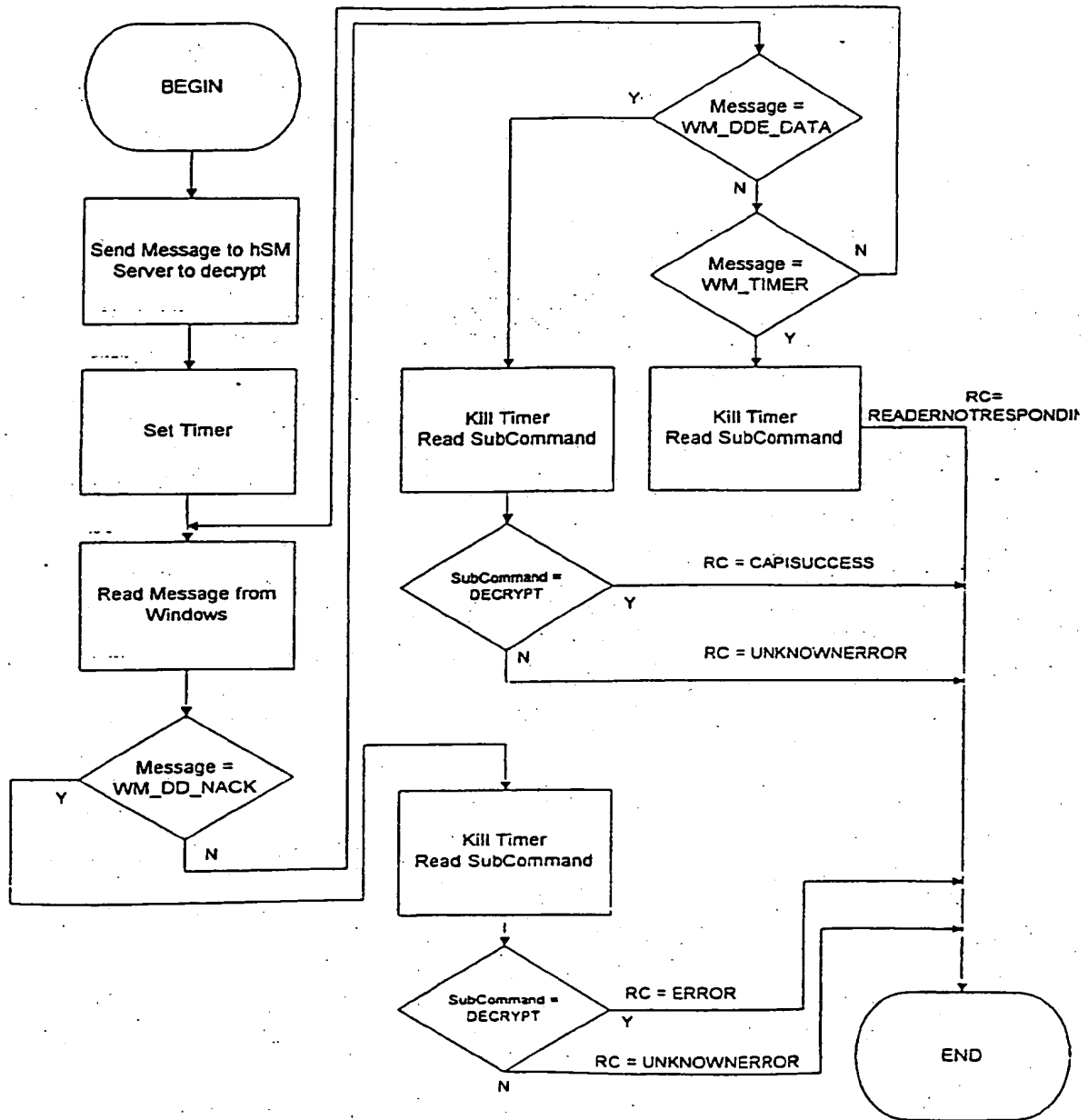
"CHART34.VSD"

CreateSecurityModuleUs r( ) Flow Chart
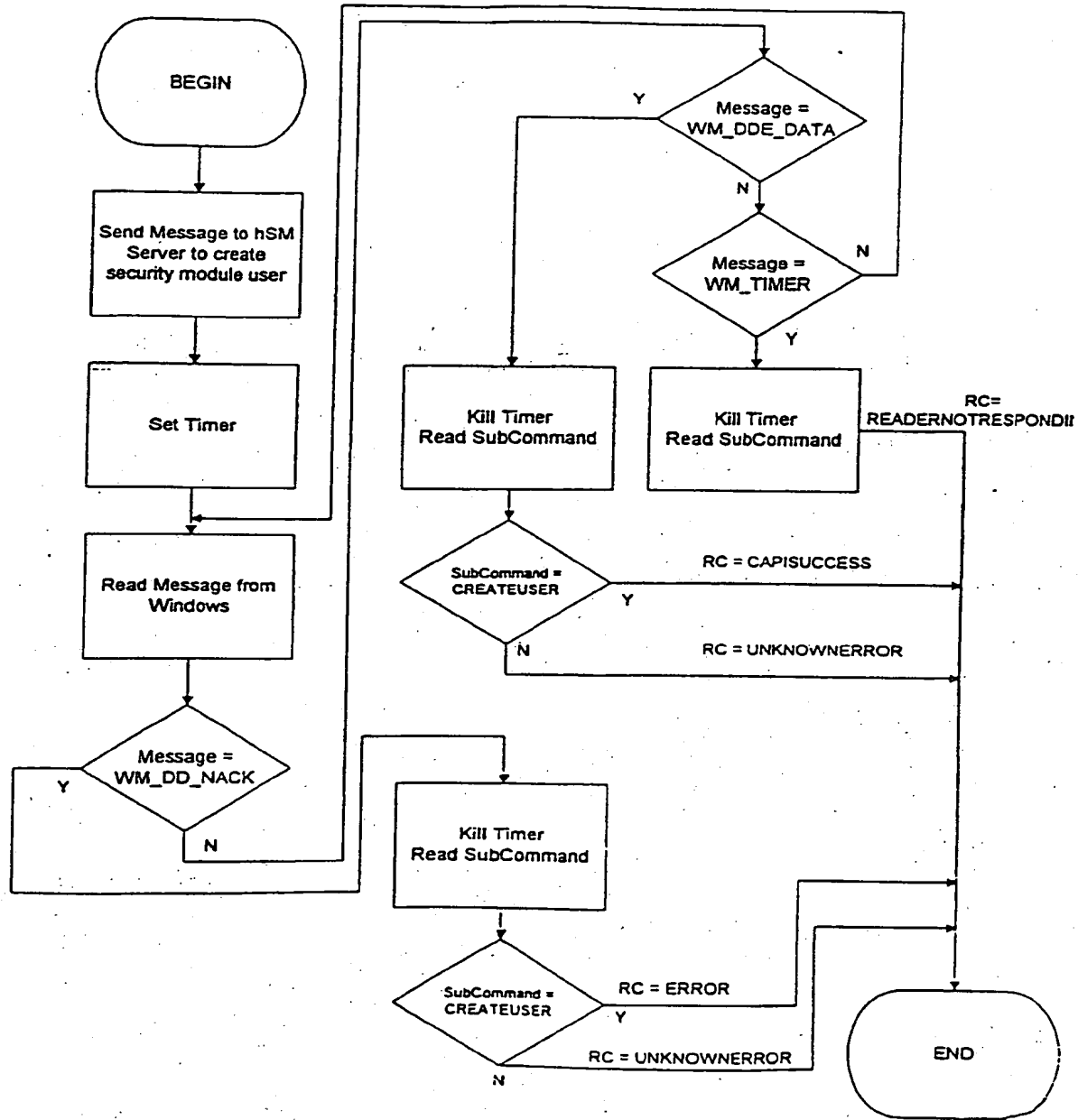
"CHART35.VSD"

DeleteSecurityModuleUser( ) Flow Chart

```
5

                    ┌─────────────┐
                   (    BEGIN      )
10                  └──────┬──────┘
                           │
                           ▼
        ┌─────────────────────────┐
        │  Send Message to hSM     │
        │  Server to delete        │
15      │  security module user    │
        └───────────┬─────────────┘
                    │
                    ▼
20      ┌─────────────────────────┐
        │       Set Timer          │
        └───────────┬─────────────┘
                    │
25                  ▼
        ┌─────────────────────────┐
        │   Read Message from      │
        │       Windows            │
        └───────────┬─────────────┘
30                  │
                    ▼
              ◇ Message =
                WM_DD_NACK ◇
35        Y                    N
```

Message = WM_DDE_DATA
  Y / N

Message = WM_TIMER
  N / Y

Kill Timer
Read SubCommand

Kill Timer
Read SubCommand   RC= READERNOTRESPOND

SubCommand = DELETEUSER
  Y — RC = CAPISUCCESS
  N — RC = UNKNOWNERROR

Kill Timer
Read SubCommand

SubCommand = DELETEUSER
  Y — RC = ERROR
  N — RC = UNKNOWNERROR

(END)

"CHART36.VSD"

Upc ₊SecurityModuleUserPassword( ) ₊ ₊w Chart

"CHART38.VSD"

PCOSCreditReq( ) Flow Chart

BEGIN

Initialize
byCommandBuf

ISOInReq(hWnd,
hReader, 17,
byCommandBuf,
wLength, byResult)

Problem to call
ISOINReq   Y   RC = return of ISOINReq

N

Result status bytes   RC = CARDERROR
= '90' '00'

N

Y

RC = CAPISUCCESS

END

"CHART50.VSC

PCOSCheckCodeReq( ) Flow Chart

```
                    ┌─────────────┐
                    │    BEGIN     │
                    └──────┬──────┘
                           │
                    ┌──────▼──────┐
                    │  Initialize  │
                    │ byCommandBuf │
                    └──────┬──────┘
                           │
                    ┌──────▼──────────┐
                    │ ISOInReq(hWnd,   │
                    │  hReader, 13,    │
                    │  byCommandBuf,   │
                    │ wLength, byResult)│
                    └──────┬──────────┘
                           │
                          ◇ ─────── Y    RC = return of ISOINReq
                    Problem to call
                      ISOINReq
                          │ N
                          │
                          ◇ ─────── RC = CARDERROR
                    Result status bytes      N
                      = '90' '00'
                          │ Y
          RC = CAPISUCCESS │
                           │
                    ┌──────▼──────┐
                    │     END      │
                    └─────────────┘
```
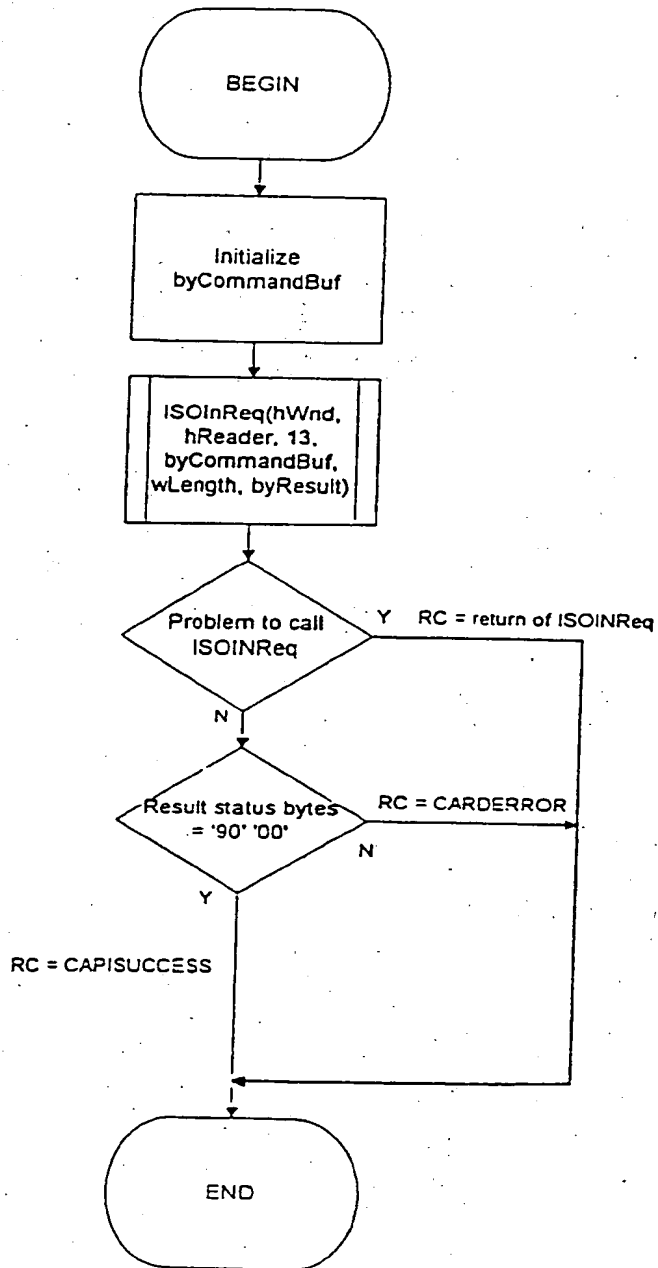
"CHART51.VSD"

PCOSDebitReq( ) Flow Chart

5

10

```
        ╭────────────╮
       │    BEGIN     │
        ╰────────────╯
              │
              ▼
       ┌──────────────┐
       │  Initialize   │
       │  byCommandBuf │
       └──────────────┘
              │
              ▼
       ┌──────────────────┐
       │ ISOInReq(hWnd,   │
       │  hReader, 13,    │
       │  byCommandBuf,   │
       │ wLength, byResult)│
       └──────────────────┘
              │
              ▼
          ◇ Problem to call ◇   Y    RC = return of ISOINReq
          ◇   ISOINReq     ◇ ──────────────────────────┐
                │                                        │
                │ N                                      │
                ▼                                        │
          ◇ Result status bytes ◇   RC = CARDERROR      │
          ◇      = '9F'        ◇ ──────────────────┐    │
                │          N                         │    │
                │ Y                                  │    │
                │                                    │    │
       RC = CAPISUCCESS                              │    │
                │                                    │    │
                ▼◄───────────────────────────────────────┘
        ╭────────────╮
       │    END       │
        ╰────────────╯
```

15

20

25

30

35

40

45

50

"CHART53.VSD"

55

47
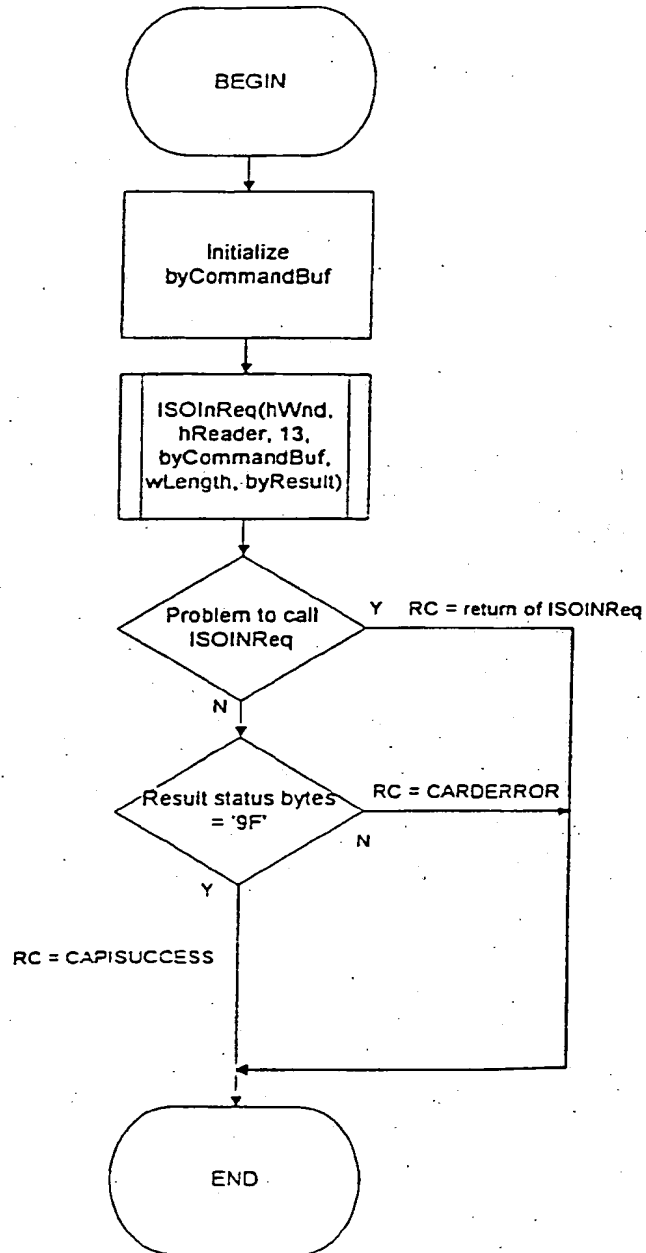
PCOSGetResponseReq( ) Flow Chart

```
                    ╭─────────────╮
                    │    BEGIN    │
                    ╰─────────────╯
                           │
                           ▼
                    ┌─────────────┐
                    │  Initialize │
                    │ byCommandBuf│
                    └─────────────┘
                           │
                           ▼
                    ┌─────────────┐
                    │ISOOUTReq(hWn│
                    │d, hReader, 5│
                    │ byCommandBuf│
                    │wLength, byResult)│
                    └─────────────┘
                           │
                           ▼
                      ╱─────────╲        Y    RC = return of ISOINReq
                     ╱ Problem to ╲──────────────────────────────┐
                     ╲ call        ╱                             │
                      ╲ ISOOUTReq ╱                              │
                       ╲─────────╱                               │
                           │ N                                   │
                           ▼                                     │
                      ╱─────────╲      RC = CARDERROR            │
                     ╱Result status╲────────────────────┐       │
                     ╲ bytes        ╱                    │       │
                      ╲ = '90' '00' ╱ N                  │       │
                       ╲─────────╱                       │       │
                           │ Y                           │       │
                           │                             │       │
              RC = CAPISUCCESS                           │       │
                           │                             │       │
                           │◄────────────────────────────┴───────┘
                           ▼
                    ╭─────────────╮
                    │     END     │
                    ╰─────────────╯
```
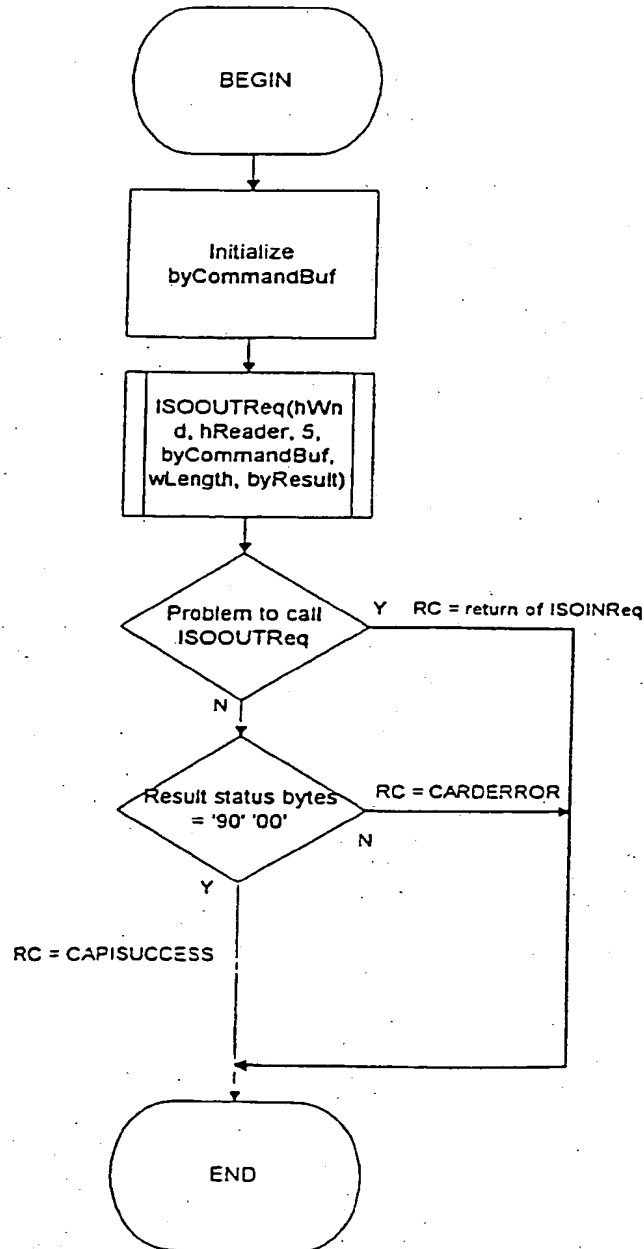
"CHART54.VSD"

PCOSReadBalanceReq( ) Flow Chart

BEGIN

Initialize
byCommandBuf

ISOINReq(hWnd,
hReader, 9,
byCommandBuf,
wLength, byResult)

Problem to call
ISOINReq — Y — RC = return of ISOINReq

N

Result status bytes
= '9F' — N — RC = CARDERROR

Y

RC = CAPISUCCESS

END
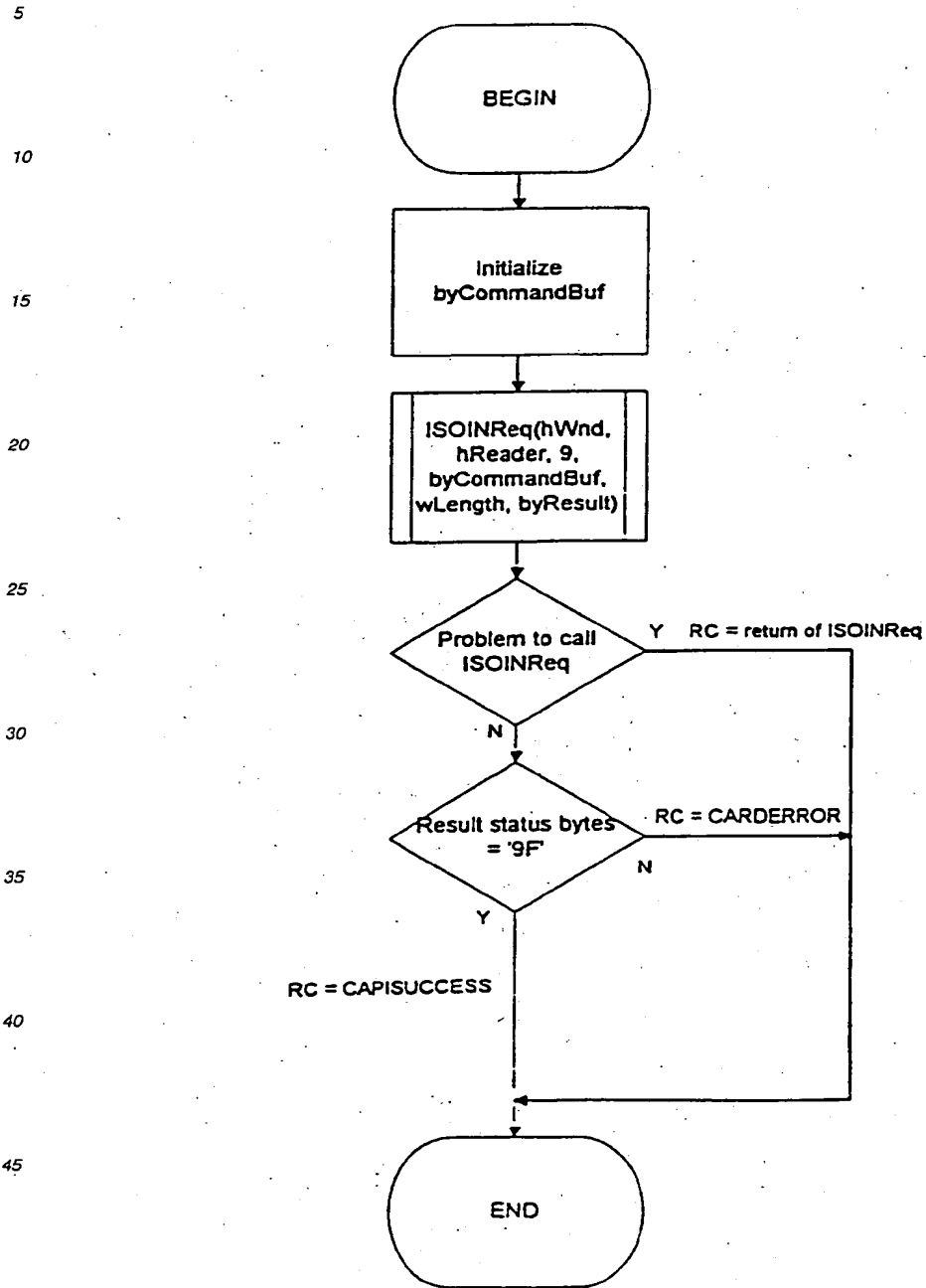
"CHART56.VSD"

PCOSReadFileReq( ) Flow Chart

```
                    ┌─────────────┐
                    │    BEGIN     │
                    └─────────────┘
                           │
                           ▼
                 ┌───────────────────┐
                 │    Initialize     │
                 │   byCommandBuf    │
                 └───────────────────┘
                           │
                           ▼
                 ┌───────────────────┐
                 │  ISOOUTReq(hWn    │
                 │  d, hReader, 5,   │
                 │  byCommandBuf,    │
                 │ wLength, byResult)│
                 └───────────────────┘
                           │
                           ▼
                       ╱───────╲                Y
                      ╱ Problem  ╲──────────  RC = return of ISOINReq
                      ╲ to call  ╱
                       ╲ISOOUTReq╱
                       ╲───────╱
                           │ N
                           ▼
                       ╱───────╲
                      ╱  Result  ╲   RC = CARDERROR
                      ╲  status  ╱────────────────►
                      ╲bytes =  ╱   N
                       ╲'90''00'╱
                       ╲───────╱
                           │ Y
         RC = CAPISUCCESS  │
                           ▼
                    ┌─────────────┐
                    │     END      │
                    └─────────────┘
```
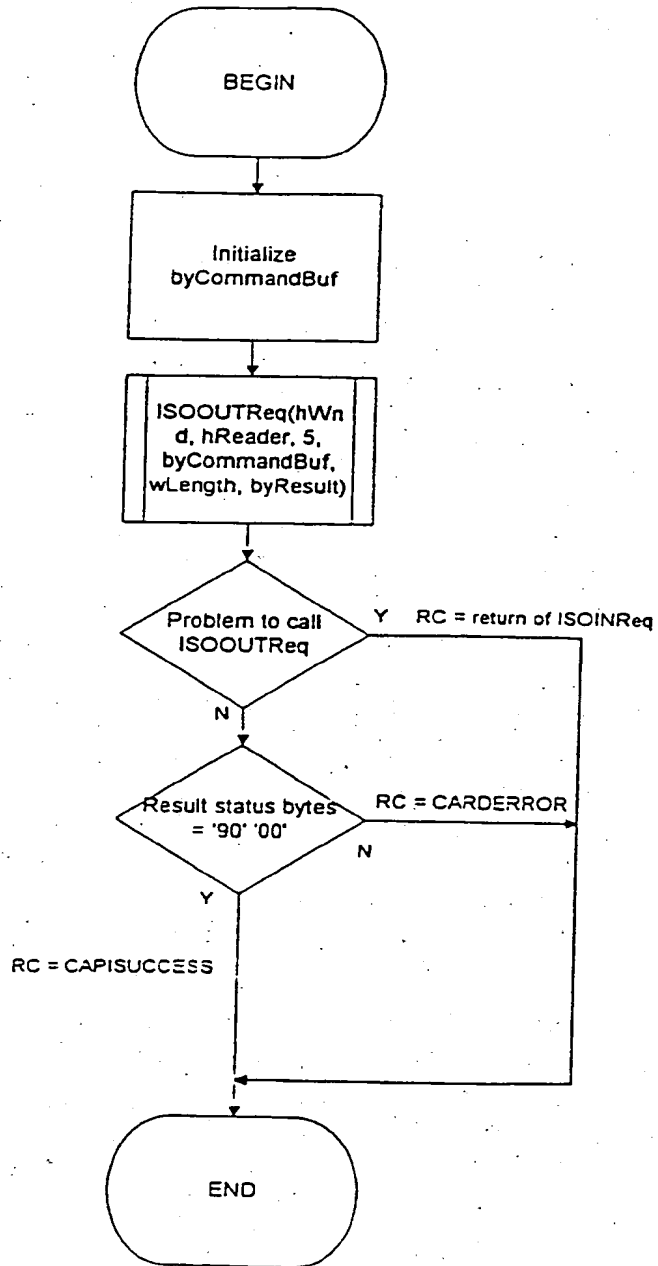
"CHART57.VSC

PCOSReadMemoryReq( ) Flow Chart
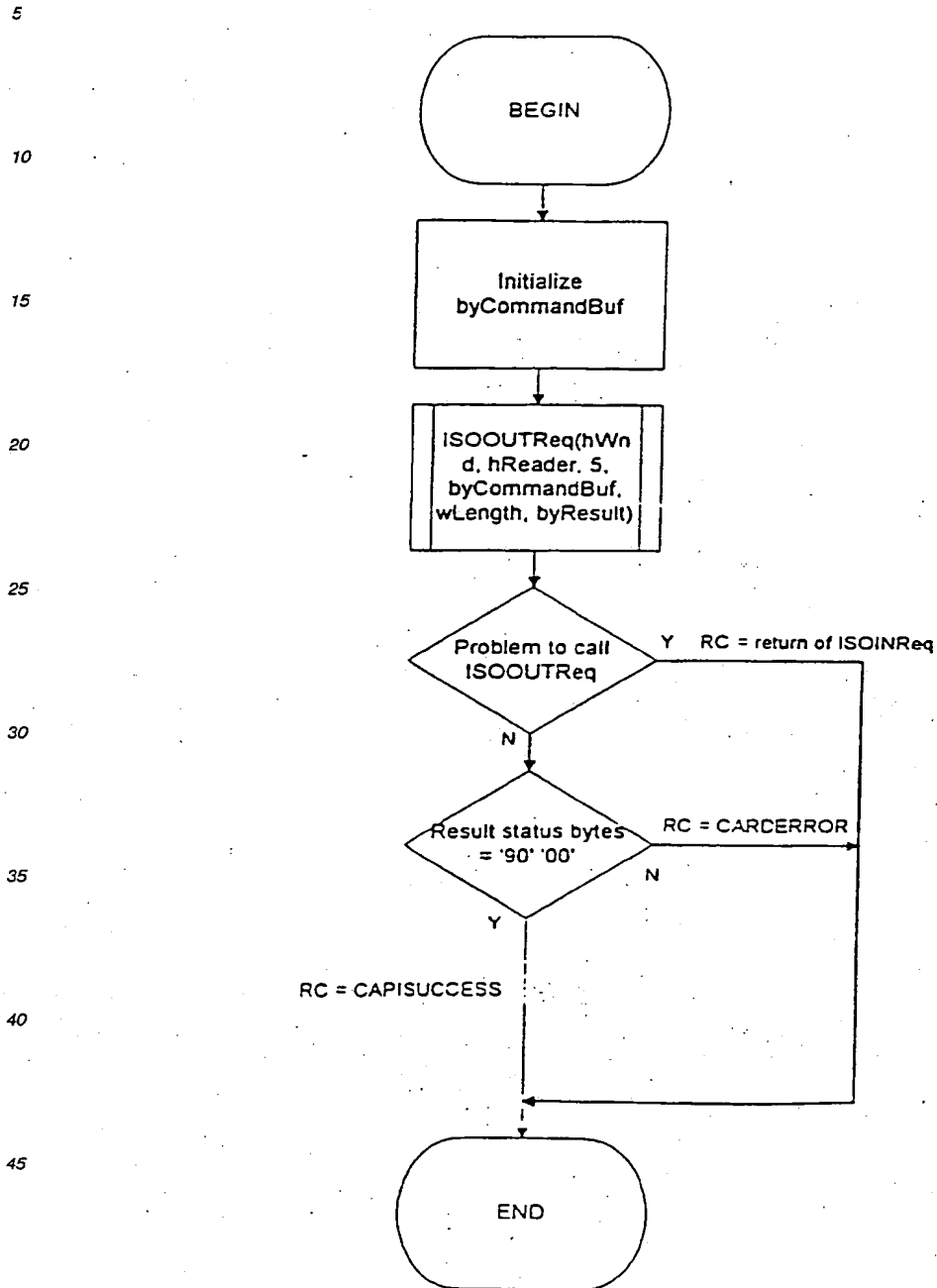
"CHART58.VSD"

PCOSSelectPurseKeyReq( ) Flow Chart

```
                    ┌─────────────┐
                    │    BEGIN     │
                    └─────────────┘
                           │
                           ▼
                    ┌─────────────┐
                    │  Initialize  │
                    │ byCommandBuf │
                    └─────────────┘
                           │
                           ▼
                    ┌──────────────────┐
                    │ ISOINReq(hWnd,   │
                    │ hReader, 15,     │
                    │ byCommandBuf,    │
                    │ wLength, byResult)│
                    └──────────────────┘
                           │
                           ▼
                       ◇ Problem to call ◇    Y    RC = return of ISOINReq
                       ◇  ISOINReq      ◇ ──────────────────────────────
                           │
                           N
                           ▼
                       ◇ Result status bytes ◇   RC = CARDERROR
                       ◇    = '9F'           ◇ ─────────────────
                           │              N
                           Y
                           │
       RC = CAPISUCCESS    │
                           ▼
                    ┌─────────────┐
                    │     END      │
                    └─────────────┘
```

"CHART59.VSD

52

PCOSSignReq( ) Flow Chart

5

10

BEGIN

15

Initialize
byCommandBuf

20

ISOOUTReq(hWn
d, hReader, 5,
byCommandBuf,
wLength, byResult)

25

Problem to call
ISOOUTReq   Y   RC = return of ISOINReq

30

N

Result status bytes   RC = CARDERROR
= '90' '00'

35

N

Y

RC = CAPISUCCESS

40

45

END

50

"CHART61.VSD"

55

53
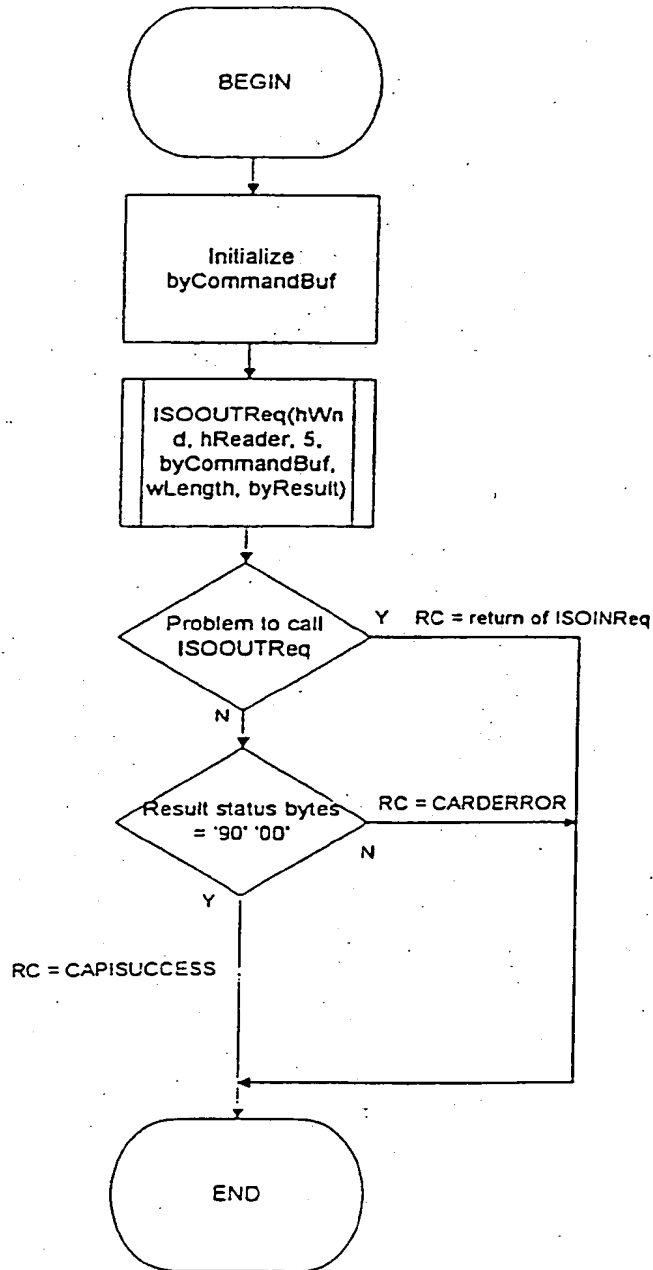
PCOSUpdateCodeReq( ) Flow Chart

```
                    ┌─────────────┐
                   (    BEGIN      )
                    └──────┬──────┘
                           │
                           ▼
                   ┌───────────────┐
                   │  Initialize   │
                   │ byCommandBuf  │
                   └───────┬───────┘
                           │
                           ▼
                  ┌─────────────────┐
                  │ ISOINReq(hWnd,  │
                  │  hReader, 13,   │
                  │  byCommandBuf,  │
                  │ wLength, byResult)│
                  └────────┬────────┘
                           │
                           ▼
                         ╱╲
                        ╱  ╲      Y    RC = return of ISOINReq
                    Problem to call ──────────────────────┐
                     ╲  ISOINReq ╱                        │
                        ╲      ╱                          │
                         ╲  ╱                             │
                          N│                              │
                           ▼                              │
                         ╱╲                               │
                        ╱  ╲   RC = CARDERROR             │
                Result status bytes ──────────────────┐   │
                    ╲ = '90' '00' ╱       N           │   │
                        ╲      ╱                       │   │
                         ╲  ╱                          │   │
                          Y│                           │   │
                           │                           │   │
         RC = CAPISUCCESS  │                           │   │
                           │                           │   │
                           ▼◄──────────────────────────┴───┘
                    ┌─────────────┐
                   (     END       )
                    └─────────────┘
```
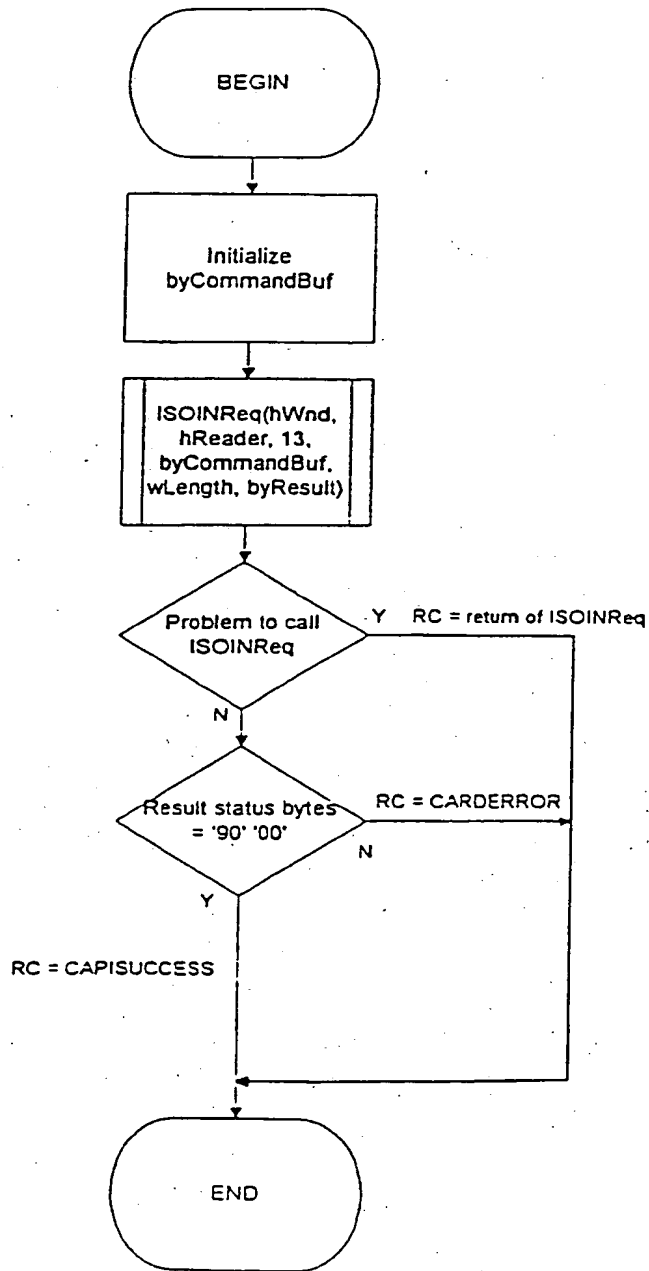
"CHART63.VSC"

PCOSUpdateFileReq( ) Flow Chart

5

10

```
         ┌─────────────────┐
        (      BEGIN        )
         └─────────────────┘
                 │
                 ▼
         ┌─────────────────┐
         │     Initialize   │
         │   byCommandBuf   │
         └─────────────────┘
                 │
                 ▼
         ┌─────────────────┐
         │ ISOINReq(hWnd,   │
         │    hReader,      │
         │   5+byLength,    │
         │  byCommandBuf,   │
         │ wLength, byResult)│
         └─────────────────┘
                 │
                 ▼
            ◇ Problem to call ◇   Y   RC = return of ISOINReq
            ◇   ISOINReq      ◇ ──────────────────────────┐
                 │                                         │
                 N                                         │
                 ▼                                         │
            ◇ Result status bytes ◇  RC = CARDERROR        │
            ◇    = '90' '00'      ◇ ────────────┐          │
                 │            N                  │          │
                 Y                               │          │
                 │                               │          │
     RC = CAPISUCCESS                            │          │
                 │                               │          │
                 ▼                               │          │
                 │◄──────────────────────────────┴──────────┘
                 ▼
         ┌─────────────────┐
        (       END         )
         └─────────────────┘
```

15

20

25

30

35

40

45

50

"CHART64 VSC

55

PCOSWriteMemoryReq( ) Flow Chart
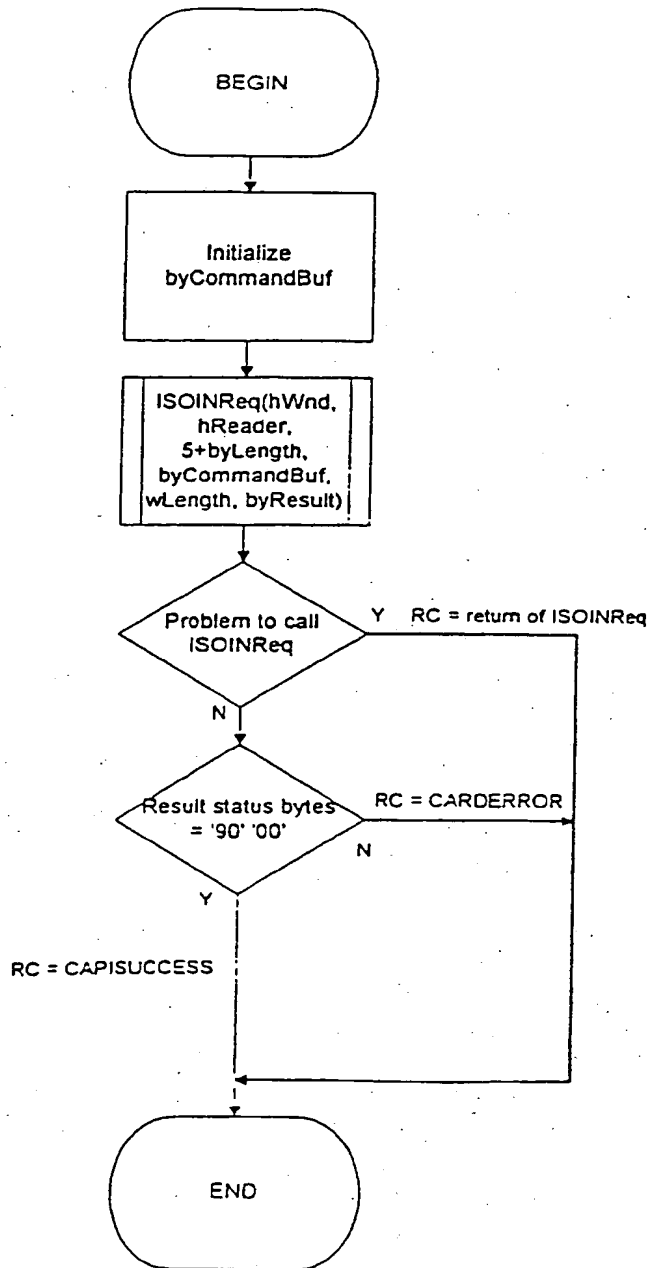
"CHART65.VSD"

PowerOnCardReq( ) Flow Chart
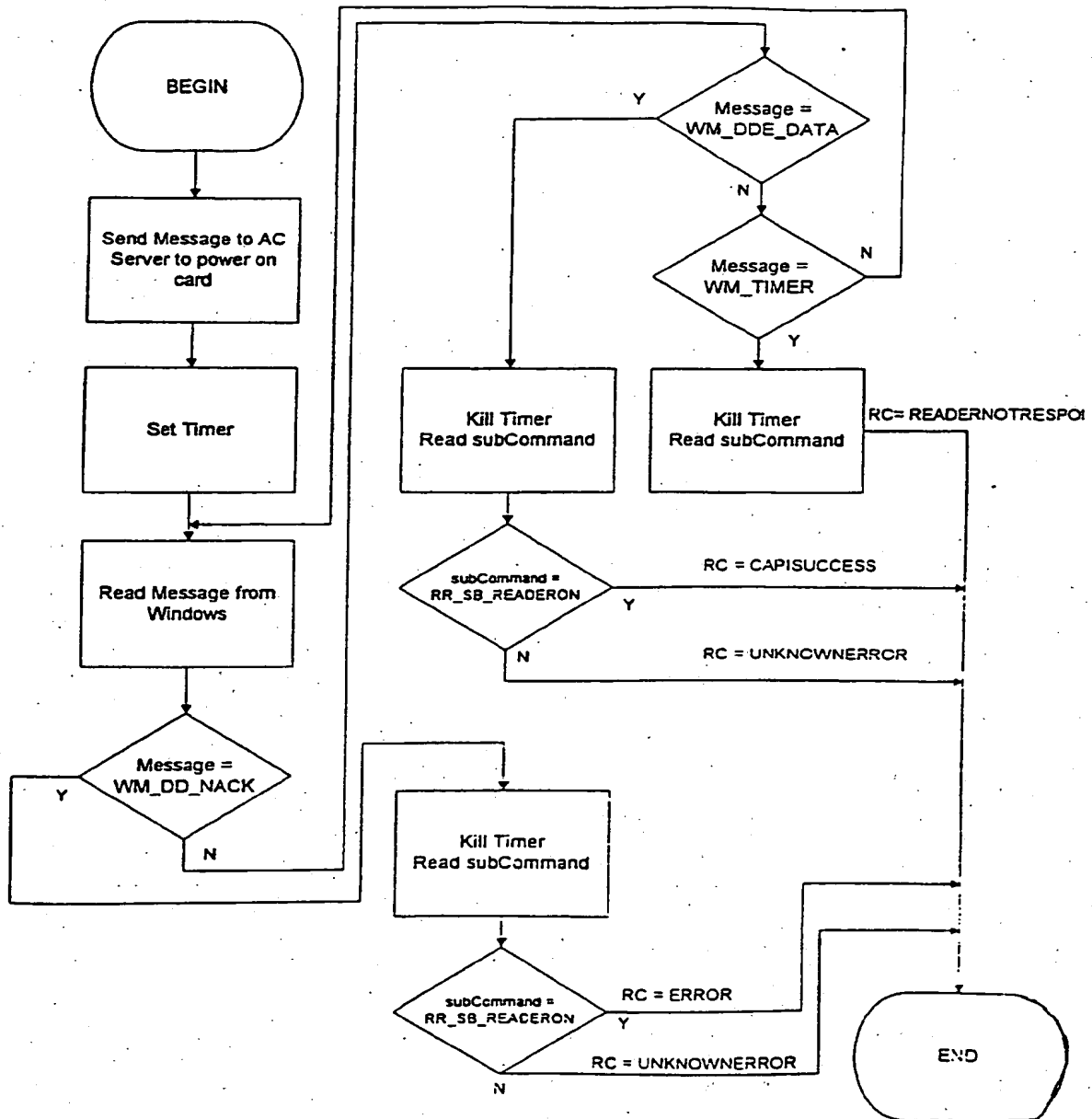
"CHART67.VSC

PowerOffCardReq( ) Flow Chart

BEGIN

Send Message to AC Server to power off card

Set Timer

Read Message from Windows

Message = WM_DD_NACK

Message = WM_DDE_DATA

Message = WM_TIMER

Kill Timer Read subCommand

Kill Timer Read subCommand

RC= READERNOTRESP

subCommand = RR_SB_READEROFF

RC = CAPISUCCESS

RC = UNKNOWNERROR

Kill Timer Read subCommand

subCommand = RR_SB_READEROFF

RC = ERROR

RC = UNKNOWNERROR

END
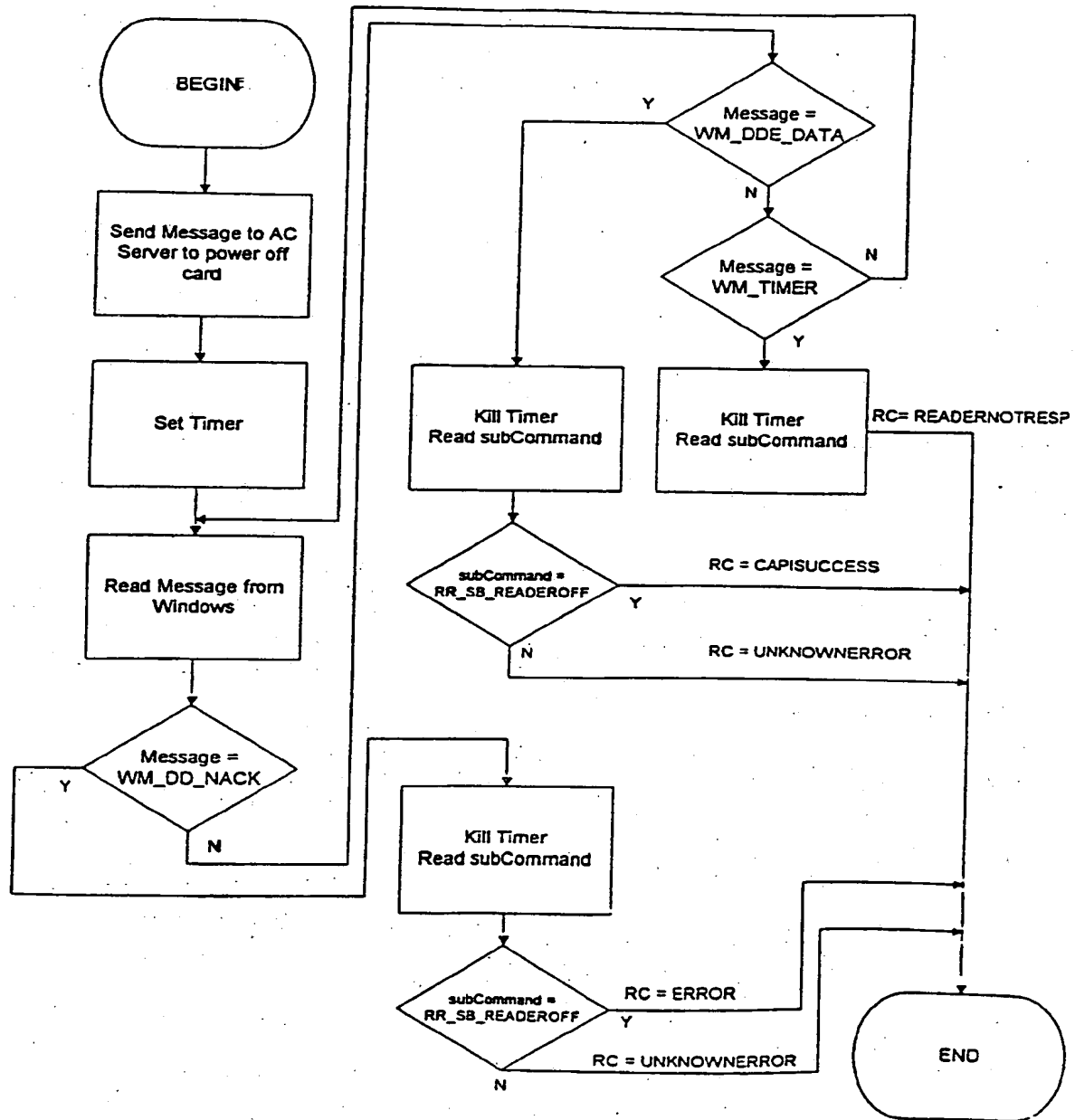
"CHART68.VSD"

ISOINReq( ) Flow Chart
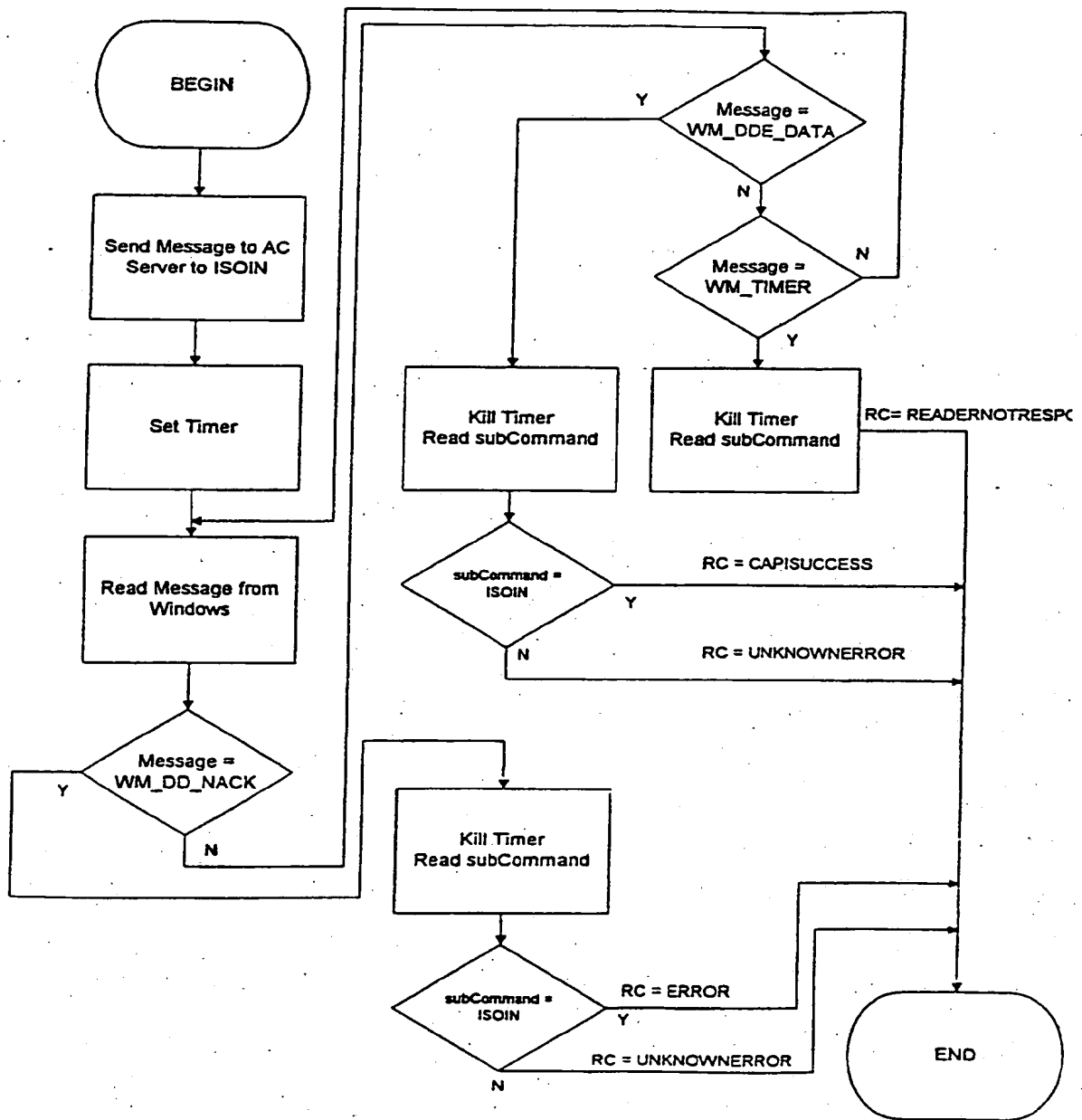
"CHART69.VSD"

ISOOUTReq( ) Flow Chart
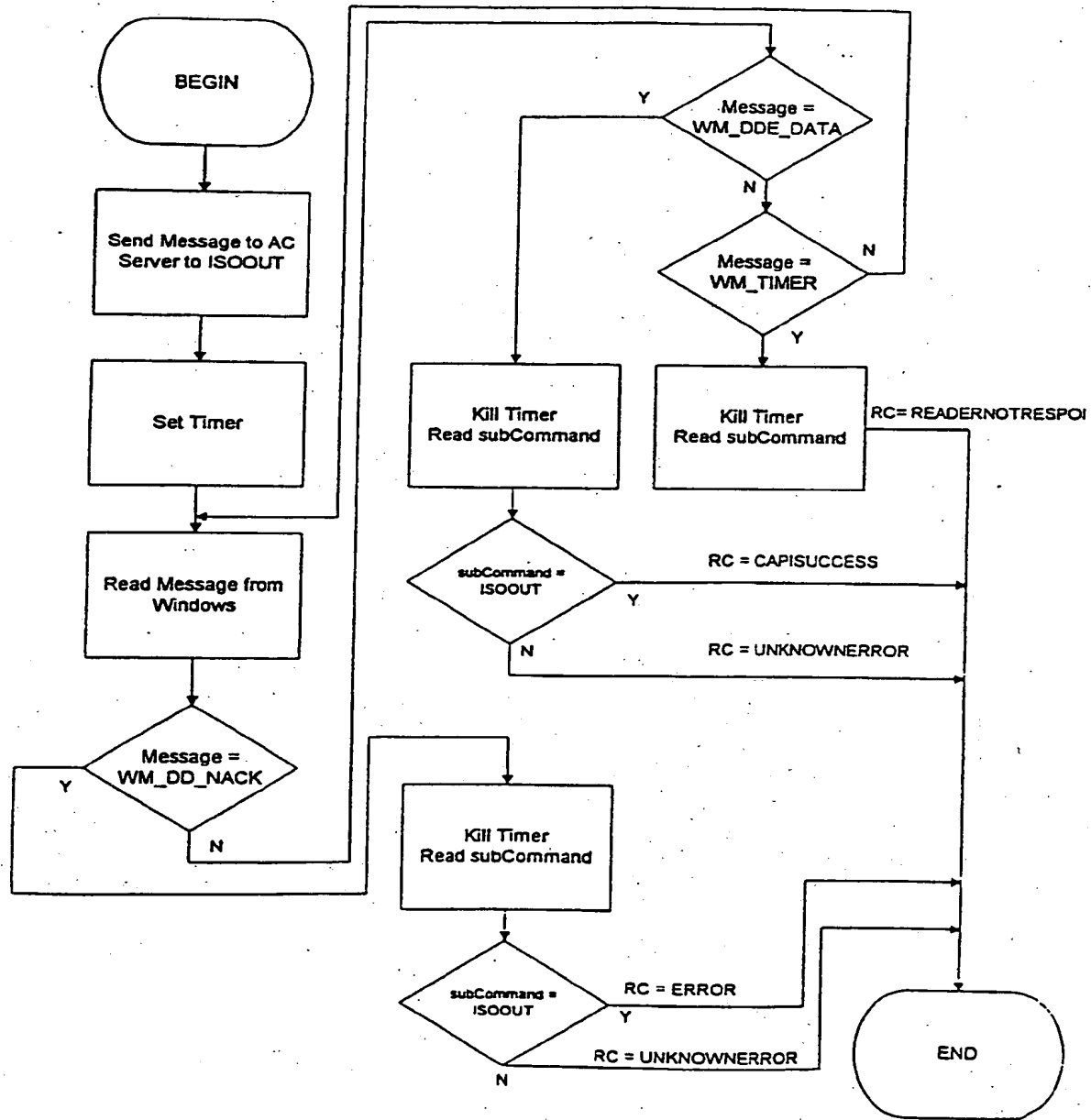
"CHART70.VSD"

APPLICATION LAYER

The application layer of CAPI is designed as the primary application programmer interface with the software and smart card hardware. It is designed to facilitate applications software operation and management of the smart card environment. CAPI provides the applications programmer with several functionalities not available on the card alone. Industry specific tool sets have been engineered to simplify interaction with lower level command sets.

The application layer of CAPI provides 23 commands. They are:

ConnectToReader

The ConnectToReader function initiates a communications session between the smart card reader and the host. It does this by requiring the ID of the reader, either COM1 or COM2, the user's name and password, and the connection mode. The connection mode indicates whether the reader will expect the password from the reader's PIN pad or directly from the host in the command parameters.

A return message is supplied to the application indicating success or failure of the command, or an error message.

DisconnectReader

This command disconnects the smart card reader from the host. The command precludes any further communication between the host and the card reader without the implementation of a valid ConnectToReader command.

The command returns a success, failure, or error message.

OpenCard

The OpenCard function initiates and resets the card. As part of this activity, the OpenCard command resets the card, reads and verifies card identification and validity information contained in the first file written on the card. This file is created upon configuration with the CAPI product. Cards not configured with the CAPI product in this manner will not allow use of the APPLICATIONS function set.

A success, failure, or error message is returned to the application as a result of this command.

ReadCardSerialNumber

ReadCardSerialNumber instructs the card reader to obtain the eight-byte serial number from the card. This information is

obtained from the first two words of the card through lower level memory reading commands.

The card serial number as well as a success, failure, or error message is returned to the application.

CloseCard

The CloseCard function powers off the card. When the CloseCard command has been used, the card will not be available until an OpenCard command is issued.

A function success, failure, or error message is produced for the application.

CreateApplication

The CreateApplication command establishes a new application for storage on the card. Two identifying and recordkeeping files are created in the card memory as a result of this command. The first of these two files contains identifying and structural information about the application. The second file is used to maintain an index of files associated with the application. The application may create up to a maximum of 60 additional files or be limited by the maximum size of the storage.

The execution of this command generates a message of success, failure, or an error.

OpenApplication

This function initiates the application and allows application operations to be performed. The application must be opened before any application related functions can be used. The application is opened when a valid application name is used.

Success, failure, and error messages are produced.

CreateUser

The CreateUser command establishes a user on the card. A total of eight users are possible on the card. These users are identified as PIN 0 to PIN 7. The CreateUser command associates a user password with one of these eight PINs. PIN 0 is reserved for the card issuer. CAPI utilizes PINs 1-7 for applications uses.

Information on success, failure, or errors is returned to the application.

Present User

The PresentUser command allows the application to offer a user name and have the card validate this name and password of this

user. The command may be executed from the reader or from the host.

The function returns a success, failure, or error message. This command permits a maximum of three invalid presentations for a user.

UpdateUser

A user's password stored on the card is changed through the use of the UpdateUser command. The application must possess the valid existing password for the user before being allowed to replace it with a new password. This information may be supplied from the host or from the card reader.

A return message of success, failure, or error type is produced.

LockUser

The UnlockUser and LockUser commands are designed to provide security for the card data for a particular user ID. The LockUser function precludes further card activity by the user and frees those resources attached to the user. The card does not have the capability to lock out a user. This lock is accomplished by the CAPI software through issuing three incorrect password presentation attempts to the card.

Success, failure, or error type messages are returned to the application.

UnlockUser

The UnlockUser command re-enables a user that has been previously locked. This function requires use of the user's password and may be executed from the card reader or host.

Error codes, as well as success and failure messages, are returned to the user, as appropriate.

CreateFile

This command allows the user to create an application file on the card. The syntax of the CreateFile command requires the user to identify the file name, size, user(s) protecting read and write access to the file, and whether the file requires ciphered access.

The card does not offer the functionality to create and maintain applications files. The CAPI software manages the raw storage to create application files and provides maintenance and indexing for these files in a way that is meaningful to the application environment.

The maximum number of files that can be created by the user is 60 or until the card storage is exhausted.

When the Create File command is used, messages are returned indicating success, failure, or error type for the function.

OpenCardFile

The OpenCardFile function opens a previously created application file on the card for use.

The application is informed of success, failure, or error type of this command.

ReadCardFile

This command permits the application software to read a stored application file that has been opened.

CAPI provides additional functionality for the application programmer not inherent to the card. The card's capability allows it to read a maximum of 32 bytes of data from memory at a time. For file sizes greater than 32 bytes, multiple read commands are required.

CAPI performs these multiple card read commands, making them transparent to the application software.

Returns are provided to the application software for function success, failure, or error type.

UpdateCardFile

UpdateCardFile allows the applications software to update previously opened files stored on the card.

The CAPI software exceeds the capabilities to the card for the application user. Card functions allow it to write a maximum of 32 bytes of information at a time in an update mode. The CAPI software accepts a single UpdateCardFile from the application and executes repeated write commands to the card until the entire update has been completed. This makes the storage management process transparent to the application program and does not limit the size of the file involved in the command.

Error, success, or failure messages are produced, as required.

CreatePurse

The CreatePurse command generates an electronic purse in the card storage. An electronic purse is a standard set of data used in the processing of financial information and transactions. It may contain balance, credit or debit, identification, and validation information. CAPI structures this data set to facilitate financial application use of the smart card environment. CAPI

dramatically simplifies the need to create, manipulate, and manage multiple files with this function.

The creation of a purse establishes three files on the card. The first file maintains the value in the purse, the second file contains account information, and the final file contains the keys related to the pursue.

The application establishes the structure of the purse, its use parameters, and validation procedures through the data supplied in the syntax of this command.

Error, success, or failure information is returned by CAPI to the application upon execution of this command.

OpenPurse

The OpenPurse command opens the purse files to the application that has the key to the purse. The command validates that the requestor has the appropriate permissions and passes this information to the security module.

Success, error-type, systemic, and functional failure messages are produced to the application, as needed.

ReadPurse

For an opened purse, the ReadPurse command provides the applications program with a data set consisting of the balance, account number, bank identity, and transactional count for the purse. CAPI performs this through accessing the multiple purse files that it has created on the card, extracting the four data elements and presenting them to the application. Return codes are also produced for the application.

DebitPurse

CAPI provides several single commands for financial transaction types. DebitPurse debits the balance contained in an opened purse. CAPI validates the debit keys provided during the OpenPurse command with the security module to ensure proper security certification. The command then returns messages to the application confirming certification of the transaction and indicating success, failure, or error type.

CreditPurse

CAPI offers the financial application user the capability to execute a single command to validate and credit the balance contained in an open purse. The CreditPurse function provides a single command to verify, and complete a credit transaction, with CAPI updating, managing, and maintaining all data and security files on the card.

CreditPurse produces a certification of the credit transaction and any associated error or completion messages of the application.

SignTransaction

The SignTransaction function provides a signature tag to an associated purse transaction. This signature provides additional authentication information to the accompanying transaction. CAPI utilizes this function to offer an additional security measure for purse transaction certification.

The command may only be used in association with a prior purse transaction. Success or failure messages are returned along with error-type identification for improper operation, use context, or syntax.

DisablePurse

DisablePurse prevents the purse from being used again. The keys associated with the purse are invalidated, making it unusable.

Message for success, failure, or error type are returned to the application.

SECURITY MODULE FUNCTIONS

The security module function set provides a separate and distinct tool kit that provides security and validation capabilities to the applications software. These functions are separated from the other applications functions so that they might reside on another physically and logically secured processor. Many of the application layer commands interact with the security module function to provide the validation and security provisions of CAPI. The functions provided by the security module are:

ConnectToSecurityModule

The ConnectToSecurityModule command establishes a communications session between the host and the software module. The command syntax requires the application to provide a valid user, password, and device identification. This information may be transmitted from the smart card reader or host.

The ConnectToSecurityModule function returns a "handle" to the application that must be used to address it in any further communications with the security module. Messages indicating success, failure, or error in establishing this connection are also returned to the application.

DisconnectSecurityModule

DisconnectSecurityModule ends the communications session between the application and the security module. The "handle" established

in the ConnectToSecurityModule function is invalidated preventing further use until reestablished through the use of another ConnectToSecurity Module command.

Success or failure of this command is indicated to the application. A single-error message denoting an improper "handle" may also be generated.

VerifyDebitCertificate

The security module interacts with the application commands processing the electronic purse transactions. This CAPI function, VerifyDebitCertificate is used to verify the debit transaction certification performed by the application command DebitPurse.

The command produces a return indicating success or failure of the verification process. It may also generate an error message if an improper "handle" is used.

VerifyCreditCertificate

The VerifyCreditCertificate is executed to validate a credit certificate from a transaction to the electronic purse. This command may be executed by the application software directly or executed by CAPI as part of a CreditPurse command.

Success or failure of the validation are confirmed to the application. An error is produced if an invalid security module "handle" is used.

CreateKey

The CreateKey command is used to create a key in the security module. A key and index is generated by the command. The index is then used to reference the key in validation and key maintenance operations. The key must be eight bytes in length and not be symmetrical.

Error, success, and failure information of this command is generated.

DeleteKey

DeleteKey removes a key from the security module. The command accesses and removes the key indicated by the key index in the command syntax.

Success, failure, or error types are returned.

SelectKey

SelectKey accesses a key in the security module indicated by the key index. This function is used for a myriad of verification and validation tasks in CAPI.

Command status or error messages are produced.

DiversifyKey

Key diversification randomizes the key utilized for the purse for each use of the card in the reader. This is designed to make prediction of the key to be used for the next session, more difficult. The user provides the data used to randomize the key. This allows the user to provide a temporary or item specific encryption schema.

Success or failure messages are produced.

SelectDiversifiedKey

This command permits the application to access a key which has been randomized by the DiversifyKey function. The user indicates the index of the key in storage, and the diversified key is furnished.

Success, failure, or error-type codes are returned to the application.

Encrypt

The Encrypt function supports the encryption of data by the security module, using the specified key. Up to eight bytes of data are encrypted and returned to the application for each command. The data is encrypted using the current selected key.

Success or failure of the encryption transaction is indicated. An error message is produced if an improper security module "handle" is used.

Decrypt

The Decrypt command decrypts data using the current selected security module key. Each command can decrypt up to eight bytes of data, previously encrypted by the security module.

RETURNS

CreateSecurityModuleUser

The command CreateSecurityModuleUser establishes a security module user and the permission set associated with that user. By using this command, the user name, and password, CreateSecurityModuleUser initiates the security module capabilities and functions available to that user. The command defines whether the user may create or delete users or keys, update users, diversify keys, encrypt date, or decrypt data.

Messages indicating if the CreateSecurityModuleUser command
succeeded or failed, or error types are generated.

DeleteSecurityModuleUser

The DeleteSecurityModuleUser function eliminates a user from
access to the security module and deletes the permission set for
that user. An application may not eliminate its own user access
nor eliminate access for any other user without appropriate
deletion privileges.

Error-type messages, as well as success of failure indications,
are made by this command.

PresentSecurity ModuleUser

This function logs a user into the security module. Using a valid
security module "handle," user name, and password,
PresentSecurityModuleUser allows an application to initiate
multiple user sessions with the security module.

Success, failure, and error-type messages are generated by this
command.

UpdateSecurityModuleUser

The UpdateSecurityModuleUser command is used to update the
password for a user of the security module. The command syntax
requires the name of the security module user, the current
password, and the new password.

The application receives a return indicating the status of the
command completion.

DISPLAY AND KEYBOARD FUNCTIONS

The display and keyboard capabilities of CAPI permit the
application to interact with video display or keyboard devices
associated with the smart card reader, if desired. These features
of CAPI provide for the maximum flexibility in device-type
utilization and represent the unique capacity of CAPI to access
the full capabilities of the read products used. The CAPI
commands available for these devices are:

GetDisplaySize

The command GetDisplaySize obtains the size of the video display
for the application. For a particular reader, the dimensions of
the display in rows and columns are returned.

In addition to the number of rows and columns, a return is
generated indicating success or failure of the operation, or error
message for use of an invalid reader "handle."

DisplayLine

This function, DisplayLine, passes a single line of text to the reader display. The command indexes the line number and provides the associated text, as part of the syntax.

Success, failure, or error-type messages are furnished.

ClearDisplay Line

The ClearDisplayLine function removes a specific line from the reader display.

Invalid line number indices or invalid reader handles produce error messages. Otherwise, the return to the application will indicate success or failure.

GetKeyBoardStatus

The GetKeyBoardStatus determines the status of the keyboard by reading of keystrokes in the keyboard buffer. If the number of keystrokes is 0, the buffer is assumed to be empty.

Success or failure of this operation is transmitted to the application. If an invalid handle is used an error message is produced.

GetKeyBoardChar

By using GetKeyBoardChar, the application reads the next character in the keyboard buffer. If no characters are present in the buffer, the return is 0.

Messages for success, failure, and improper reader handles are also returned to the application.

GetKeyBoardString

The application may read the entire contents of the keyboard buffer at one time. This is done using the command GetKeyBoardString. All data in the keyboard buffer is then supplied to the application.

Success, failure, and an error message for an invalid reader handle may also be suppled to the application.

ClearKeyBoardBuffer

ClearKeyBoardBuffer deletes all information currently held in the reader keyboard buffer.

Command success or failure is indicated to the application. A single error message indicating an invalid reader handle may also be produced.

GetPIN

This CAPI function obtains four-byte PIN number from the reader keyboard and returns it to the application. Along with the PIN number, a message describing command success or failure, or invalid reader handle error, is returned to the application.

CARD LAYER

The card layer of CAPI functionality provides the application with direct access to the card functions.

CreditReq

CreditReq command provides application access to the card level purse credit function. CreditReq supplies the card level command with the credit amount and credit key data. Error messages success or failure of the command are returned to the application.

CheckCodeReq

The CheckCodeReq validates a PIN number stored on the card. Up to eight codes may be stored on the card, designated PIN 0 through PIN 7. PIN 0 is reserved for the card issuer. The CheckCodeReq command compares the designated PIN to the code stored.

Success or failure of the comparison, as well as error code type messages are produced.

CheckKeyReq

The CheckKeyReq function enables an application to confirm a DES key. This command checks the key storage on the card to confirm the identity of the DES key file to be checked.

A return of CAPISUCCESS indicates that the DES key file is appropriate for the key file index provided. Failure and error-type messages are also produced.

DebitReq

DebitReq supports direct access to the card level debit purse function. Success, failure, and error-type messages are issued.

GetResponseReq

The GetResponseReq function manages the query and response structure between the reader and the card. It structures the card query of reader in terms of the expected response length, actual

response length, and respondent data. The nature of the data provided by these queries is dependent upon the context of use with other commands.

Success, failure, and error messages are produced for this structuring command.

InvalidateKeyReq

This function permits the application direct access to the card capability to invalidate a DES key.

As with all other CAPI commands, success, failure, and error messages are produced for the application.

ReadBalanceReq

The ReadBalanceReq command utilizes the card capabilities to read the balance of the selected electronic purse. This CAPI function provides the card the indexing information needed to read the appropriate purse. The actual balance data is obtained through the use of the GetResponseReq command.

Success, failure, and error messages are generated.

ReadFileReq

ReadFileReq reads data from a card file. For an indexed file, this function can provide up to 32 bytes of data to the application from files stored on the card.

Messages are supplied to the application indicating success, failure, or error type.

ReadMemoryReq

The ReadMemoryReq function allows the application to directly interrogate specific memory addresses on the card. The application provides a high and low memory address and up to 32 bytes of the data contained in the memory range is returned. Providing a range of over 32 bytes for a single command will result in an error.

Success, failure, and error messages, as well as the actual length of the data read and read data, is supplied to the application.

SelectPurseKeyReq

SelectPurseKeyReq accesses a purse key stored on the card in association with credit or debit transactions. Success or failure of this operation indicates whether the key is confirmed or not. This command also produces error messages, when appropriate.

SetOptionsReq

The CAPI SetOptionsReq command allows the application to set two card options. The first option allows the application to determine if the current electronic purse's balance should be used in the debit or credit certificate calculation. The second option permits the application software to clear the parameters stored in RAM to be cleared after the SignReq command has been executed. Both of these options are selected by setting a single bit flag in the command syntax.

Success, failure, or error-type messages are produced.

SignReq

The SignReq command elicits signature data from the card for a preceding electronic purse transaction. This is part of the validation process for electronic purse credit and debit transactions.

The electronic signature data, and a message indicating success, failure, or error-type, are returned to the application.

SubstituteDebitReq

This CAPI function accesses the card capability to cancel a preceding electronic purse debit transaction and substitute a new debit transaction.

A message indicating success, failure, or nature of error is returned to the application.

UpdateCodeReq

The UpdateCodeReq command permits the application to change one of the eight PIN codes stored on the card. The new eight byte PIN code is then maintained on the card.

Success, failure or error type are all identified to the application.

UpdateFileReq

The UpdateFileReq function allows the CAPI user to change data in an existing file. Up to 32 bytes of data at a time may be modified in any one of the application files stored on the card.

The operation generates a message describing whether it has succeeded, failed or encountered an error.

WriteMemoryReq

CAPI provides direct access to the memory address. WriteMemoryReq writes data to a specific storage address range. Each command may write up to 32 bytes of information.

The application is informed of the success, failure or error type resulting from this command.

ISO LAYER

The ISO layer of the CAPI commands allows the application program to control the basic operation and connectivity of the SMAC configuration.

ConnectToReadReq

The ConnectToReadReq establishes communication between an application program and a device in the SMAC configuration. Upon confirmation of appropriate user name and password information, the application is permitted to connect to the reader. This may be accomplished from the host or another reader.

A message is generated describing success, failure, or error type for this operation.

PowerOnCardReq

This CAPI command turns the power on for the card from the reader. The function resets the card and provides the reset response data to the application for confirmation. A success, failure, or error-type is reported to the application, as well.

As with all CAPI commands, success, failure, or error-type are communicated to the application.

PowerOffCardReq

PowerOffCardReq allows the application to turn the power off for the card from the reader. Confirmation of this command is provided by the return of "CAPISUCCESS." Messages are also generated for command failure or errors in execution.

ISOINReq

This CAPI feature transmits an ISOIN command to the card. CAPI passes this ISO command to the card and returns the card response to the application and the attached reader.

Errors, success, or failure are returned to the application, as well.

ISOOUTReq

ISOOUTReq allows the application to send an ISOOUT command to the card. This command conveys the ISO command to the card and returns the response.

Success, failure, or errors incurred with command execution, in addition to the ISOOUT card response are returned.

Function Name	Location	Flow Chart File Name	Comment
ConnectToReaderReq( )	ISO.DLL	CHART1.VSD	
DisconnectReaderReq( )	ISO.DLL	CHART2.VSD	
PCOpenCard( )	PCCAPI.DLL	CHART3.VSD	
PCReadCardSerialNumber( )	PCCAPI.DLL	CHART4.VSD	
PCCloseCard( )	PCCAPI.DLL	CHART5.VSD	
PCCreateApplication( )	PCCAPI.DLL	CHART6.VSD	
PCOpenApplication( )	PCCAPI.DLL	CHART7.VSD	
PCCreateUser( )	PCCAPI.DLL	CHART8.VSD	
PCPresentUser( )	PCCAPI.DLL	CHART9.VSD	
PCUpdateUser( )	PCCAPI.DLL	CHART10.VSD	
PCLockUser( )			Not available
PCUnlockUser( )			Not available
PCCreateFile( )	PCCAPI.DLL	CHART13.VSD	
PCOpenCardFile( )	PCCAPI.DLL	CHART14.VSD	
PCReadCardFile( )	PCCAPI.DLL	CHART15.VSD	
PCUpdateCardFile( )	PCCAPI.DLL	CHART16.VSD	
PCCreatePurse( )	PCCAPI.DLL	CHART17.VSD	
PCOpenPurse( )	PCCAPI.DLL	CHART18.VSD	
PCReadPurse( )	PCCAPI.DLL	CHART19.VSD	
PCDebitPurse( )	PCCAPI.DLL	CHART20.VSD	
PCCreditPurse( )	PCCAPI.DLL	CHART21.VSD	
PCSignTransaction( )			Not available
PCDisablePurse( )			Not available
ConnectSecurityModule( )	SM.DLL	CHART24.VSD	
DisconnectSecurityModule( )			Not available
VerifyDebitCertificate( )	SM.DLL	CHART26.VSD	
VerifyCreditCertificate( )	SM.DLL	CHART27.VSD	
CreateKey( )	SM.DLL	CHART28.VSD	
DeleteKey( )	SM.DLL	CHART29.VSD	
SelectKey( )	SM.DLL	CHART30.VSD	
DiversifyKey( )	SM.DLL	CHART31.VSD	
SelectDiversifiedKey( )			Not available
Encrypt( )	SM.DLL	CHART33.VSD	
Decrypt( )	SM.DLL	CHART34.VSD	
CreateSecurityModuleUser( )	SM.DLL	CHART35.VSD	
DeleteSecurityModuleUser( )	SM.DLL	CHART36.VSD	
PresentSecurityModuleUser( )			Not available
UpdateSecurityModuleUser( )	SM.DLL	CHART38.VSD	
GetDisplaySize( )			Not available
DisplayLine( )			Not available
ClearDisplayLinee( )			Not available
GetKeyBoardStatus( )			Not available

GetKeyBoardChar( )			Not available
GetKeybBoardString( )			Not available
ClearKeyBoardBuffer( )			Not available
GetPIN( )			Not available
PCOSCreditReq( )	PCOS.DLL	CHART50.VSD	
PCOSCheckCodeReq( )	PCOS.DLL	CHART51.VSD	
PCOSCheckKeyReq( )			Not available
PCOSDebitReq( )	PCOS.DLL	CHART53.VSD	
PCOSGetResponseReq( )	PCOS.DLL	CHART54.VSD	
PCOSInvalidateKeyReq( )			Not available
PCOSReadBalanceReq( )	PCOS.DLL	CHART56.VSD	
PCOSReadFileReq( )	PCOS.DLL	CHART57.VSD	
PCOSReadMemoryReq( )	PCOS.DLL	CHART58.VSD	
PCOSSelectPurseKeyReq( )	PCOS.DLL	CHART59.VSD	
PCOSSetOptionsReq( )			Not available
PCOSSignReq( )	PCOS.DLL	CHART61.VSD	
PCOSSubstituteDebitReq( )			Not available
PCOSUpdateCodeReq( )	PCOS.DLL	CHART63.VSD	
PCOSUpdateFileReq( )	PCOS.DLL	CHART64.VSD	
PCOSWriteMemoryReq( )	PCOS.DLL	CHART65.VSD	
ConnectToReaderReq( )			Not available
PowerOnCardReq( )	ISO.DLL	CHART67.VSD	
PowerOffCardReq( )	ISO.DLL	CHART68.VSD	
ISOINReq( )	ISO.DLL	CHART69.VSD	
ISOOUTReq( )	ISO.DLL	CHART70.VSD	
GeneralCommandReq( )	ISO.DLL	CHART80.VSD	

Claims

1. A system for controlling requests from portable electronic cards of differing in-card electronic processing capabilities, and including a central processing module, associated memory, and an operating system, comprising:

   a plurality of card units into which data cards and cards having internal data processing may be inserted;
   security module means for authenticating cards inserted into the plurality of card units and for securely exchanging data with the authenticated cards; and
   means, separate from the security module means, for processing data received from the security module in accordance with an application program.

2. The system according to claim 1, wherein said security module comprises means for providing data security for multiple applications.

3. The system according to claim 1 or claim 2, wherein the processing means comprises an application module, the application module including:

   means for interfacing a plurality of differing data carrying cards with the application module.

4. The system according to any of claims 1 to 3, wherein the processing means comprises:

   means for transferring data to and from files that relate to an application program of a particular card through the security module.

5. The system according to any of the preceding claims, wherein the security module comprises:

means for validating a plurality of properties of the card, including validating an internal key.

6. The system according to any of the preceding claims, wherein the security module includes a secure microprocessor for encrypting and decrypting data.

7. The system according to claim 6, wherein the security module includes a first secure memory for securely storing encryption keys.

8. The system according to claim 6 or claim 7, wherein the security module includes a second secure memory for securely storing a security program.

9. A method of securely exchanging data between a data-carrying card and a processing system, comprising the steps of:

   validating the authenticity of a data-carrying card at a security module of the processing system;
   providing data from the authenticated data-carrying card to the security module of the processing system;
   processing the data at the security module;
   providing the processed data from the security module to an application module; and
   processing the data at the application module.

10. The method according to claim 9, wherein the step of validating the authenticity of the data-carrying card includes the steps of:

    providing a random number in the processing system;
    transmitting the random number from the processing system to the data-carrying card;
    encrypting the random number at the data-carrying card using a first internal key stored in the card;
    providing the encrypted number from the card to the processing system;
    decrypting the encrypted number at the processing system using a second internal key stored in the processing system;
    comparing the decrypted number with the random number in the processing system; and
    determining that the data-carrying card is authentic if the decrypted number and the random number in the processing system match.

11. The method according to claim 9 or claim 10, further comprising the step of validating the authenticity of the processing system at the data-carrying card.

12. The method according to claim 11, wherein the step of validating the authenticity of the processing system includes the steps of:

    providing a random number in the data-carrying card;
    transmitting the random number from the card to the processing system;
    encrypting the random number at the processing system using a first internal key stored in the card;
    providing the encrypted number from the processing system to the card;
    decrypting the encrypted number at the card using a second internal key stored in the card;
    comparing the decrypted number with the random number in the card; and
    determining that the processing system is authentic if the decrypted number and the random number in the card match.

13. The method according to any of claims 9 to 12, further comprising the steps of:

    verifying that a user is authorized to use the card.

14. The method according to any of claims 9 to 13, wherein the step of processing the data at the security module includes the step of decrypting the data using decryption keys.

15. The method according to any of claims 9 to 14, further comprising the steps of:

    providing the data from the application module to the security module;
    processing the data at the security module; and

storing the processed data on the data card.

16. The method according to claim 15, wherein the step of processing the data at the security module includes the step of encrypting the data using encryption keys.

17. A system for providing secure exchange of data with data-carrying cards, comprising:

a first processor programmed to encrypt and decrypt data, the first processor being a secure processor;
a secure memory connected to the secure processor for storing a security program and encryption and decryption keys;
a second processor programmed to execute an application program in accordance with data received from the first processor,

18. The system according to claim 17, further comprising an input/output interface circuit provided to interface a card reader with the first processor.

19. The system according to claim 17 or claim 18, further comprising a host processor connected to the second processor and programmed to provide system functions.

20. A security module, comprising:

an input/output interface from which data can be received and transmitted;
data-encryption means for encrypting data in accordance with an encryption technique using encryption keys;
a memory for securely storing the encryption keys; and
means for securely managing the encryption keys stored in the memory.

21. The security module according to claim 20, further comprising a card reader connected to the input/output interface for reading data from data-carrying cards.

22. The security module according to claim 21, further comprising means for authenticating cards inserted into the card reader.

23. The security module according to claim 21 or claim 22, further comprising means for authenticating the security module to cards inserted into the card reader.

24. The security module according to any of claims 21 to 23, further comprising means for verifying that a user is authorized to use a card inserted into the card reader.

25. The security module according to any of claims 20 to 24, further comprising means for authenticating data received from the input/output interface.
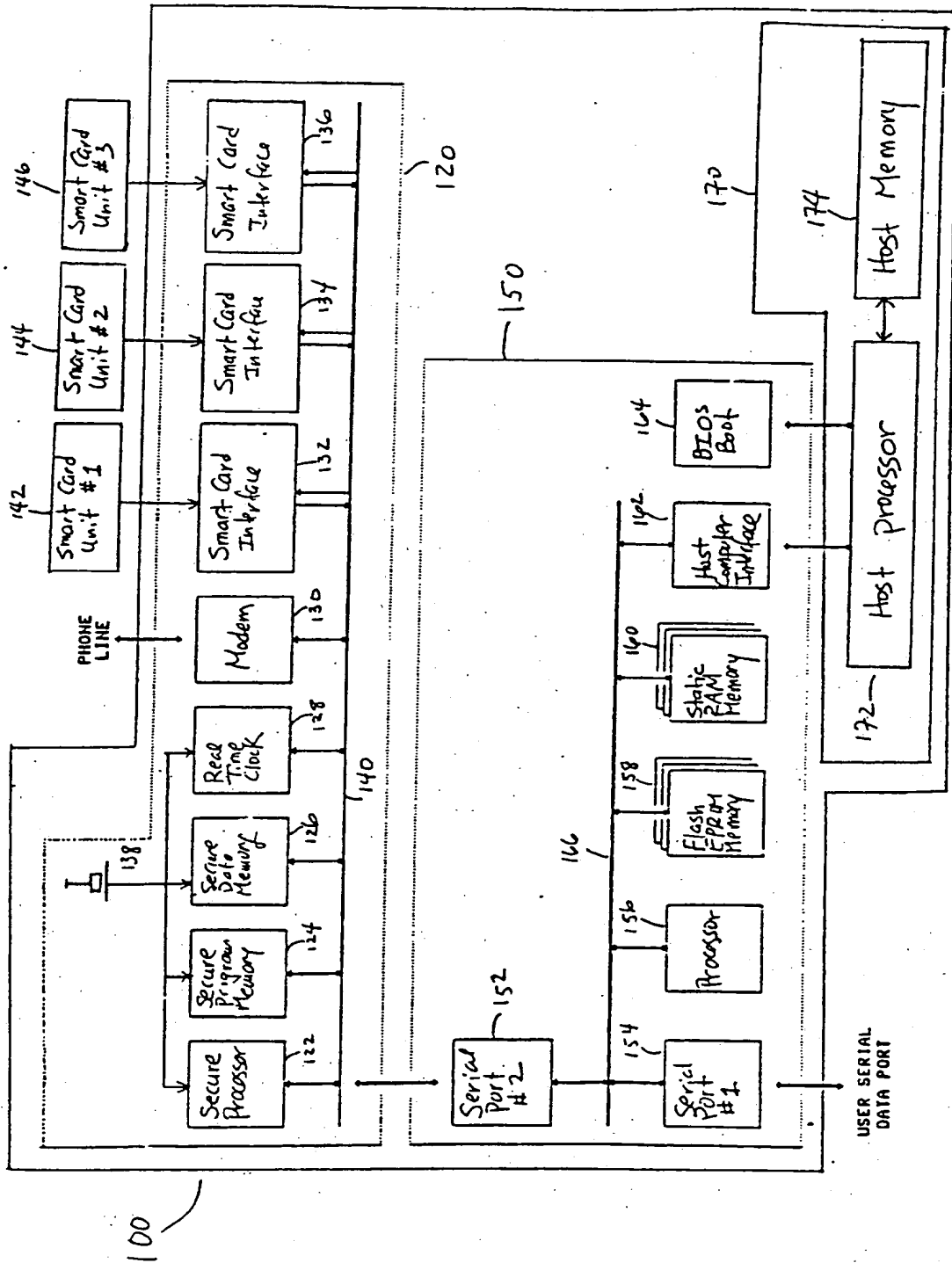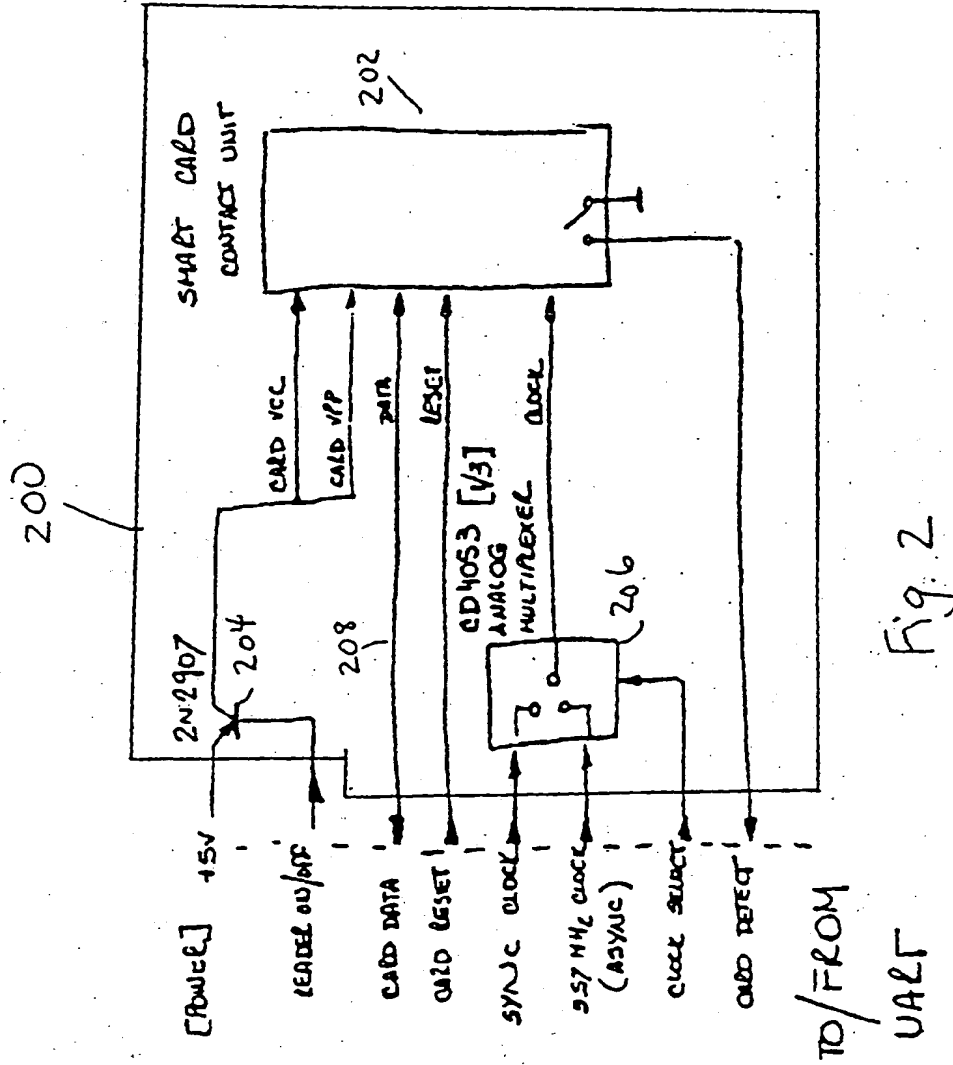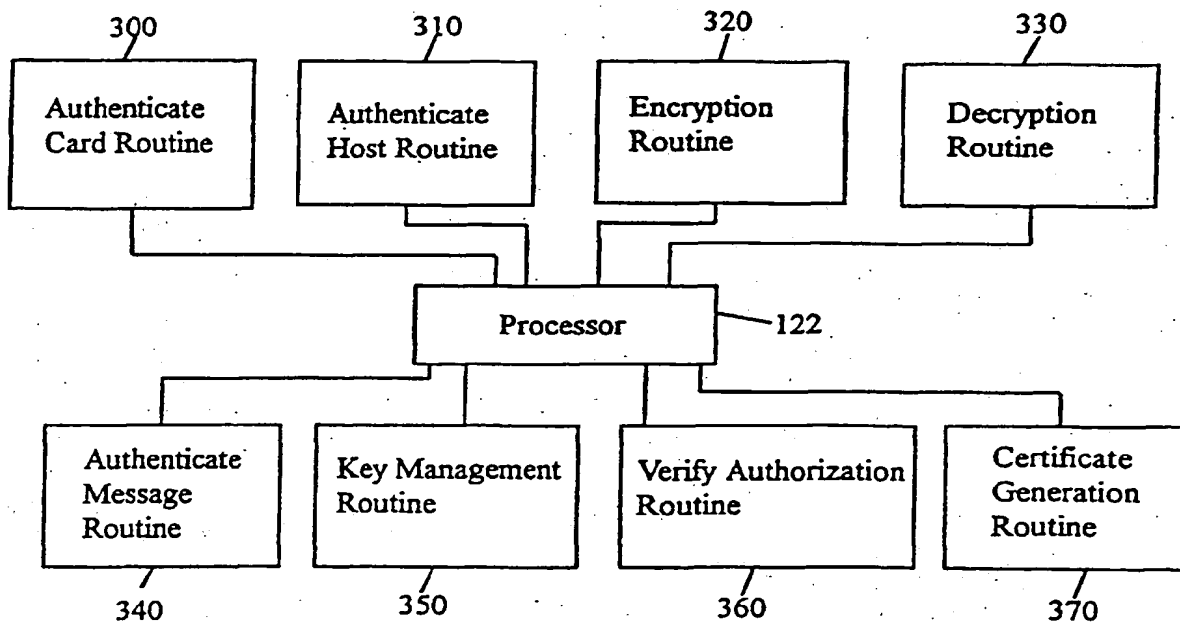
Figure 1

SMART CARD CONTACT UNIT

202

200

CARD VCC
CARD VPP
DATA
RESET
CLOCK

CD4053 [V3]
ANALOG MULTIPLEXER

206

204

2N2907

208

[POWER] +5V

READER ON/OFF

CARD DATA

CARD RESET

SYNC CLOCK

3.57 MHz CLOCK (ASYNC)

CLOCK SELECT

CARD DETECT

TO/FROM UART

Fig. 2

Fig. 3

Application Program 400

CAPI

Application Layer 410
412

Card Layer 414
Process

ISO Layer 416

CAD (Card Acceptance Device)

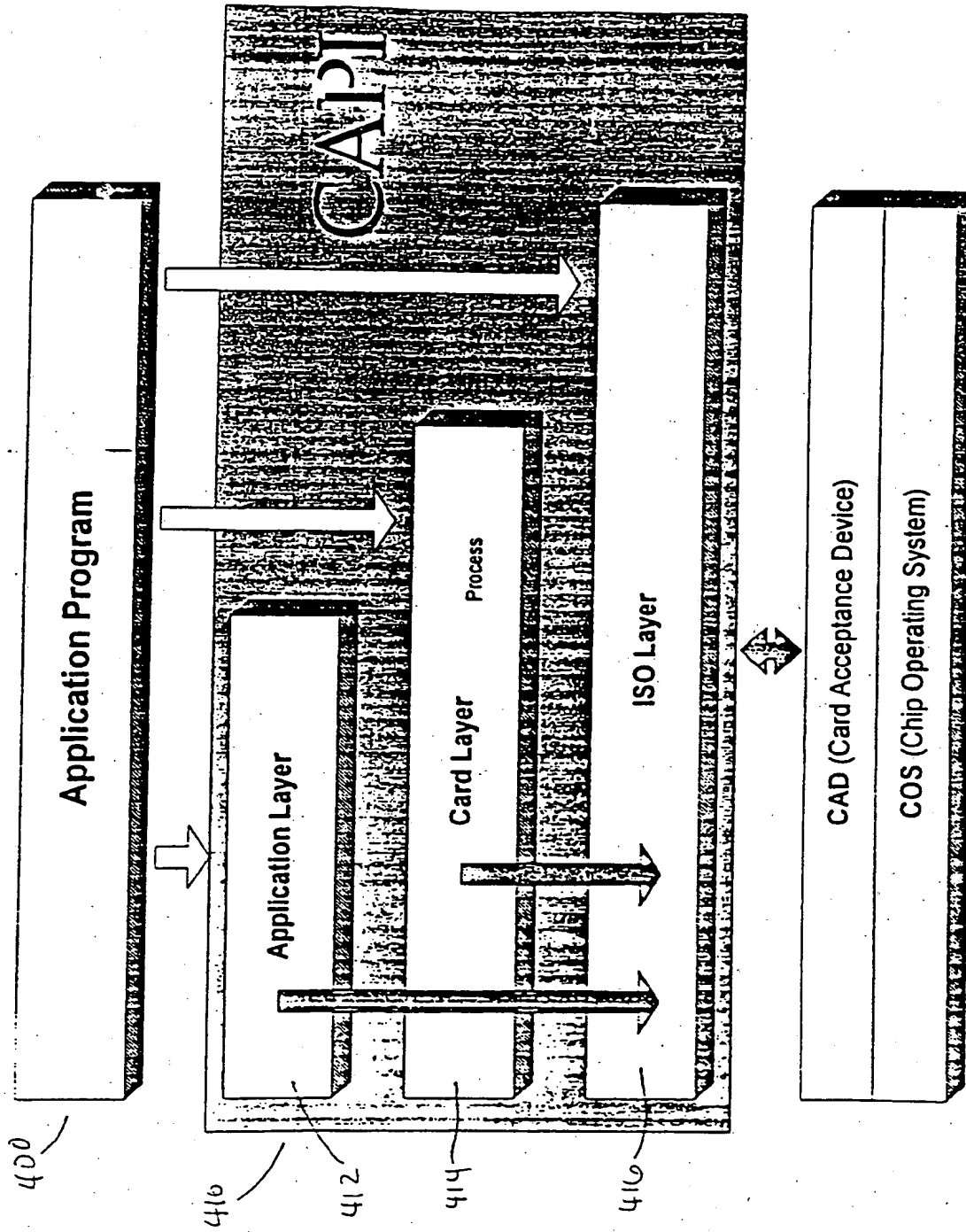**COS (Chip Operating System)**

Fig. 4

European Patent
Office

EUROPEAN SEARCH REPORT

Application Number

EP 96 30 3319

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
Y	US 5 036 461 A (J.C. ELLIOTT)	1,9,17, 20	G07F7/10 G06K7/06
A	* abstract; claims; figures *	3-8, 10-16, 18,19, 21-25	
	* column 8, line 14 - column 12, line 7 *		
Y	EP 0 456 548 A (DASSAULT ÉLECTRONIQUE)	1,9,17, 20	
A	* abstract; claims; figure 3 *	2-8,13, 14,16, 18,19, 21,22, 24,25	
A	EP 0 355 372 A (SPA SYSPATRONIC)	1,3-9, 13-25	
	* abstract; claims; figures * * column 4, line 18 - column 5, line 51 *		
A	WO 96 13791 A (SIEMENS)		TECHNICAL FIELDS SEARCHED (Int.Cl.6)
A	WO 95 04328 A (INTELLECT AUSTRALIA)		G07F G06K
A	EP 0 348 932 A (OKI ELECTRIC INDUSTRY)		

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	6 December 1996	David, J

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EPO FORM 1503 03.82 (P04C01)